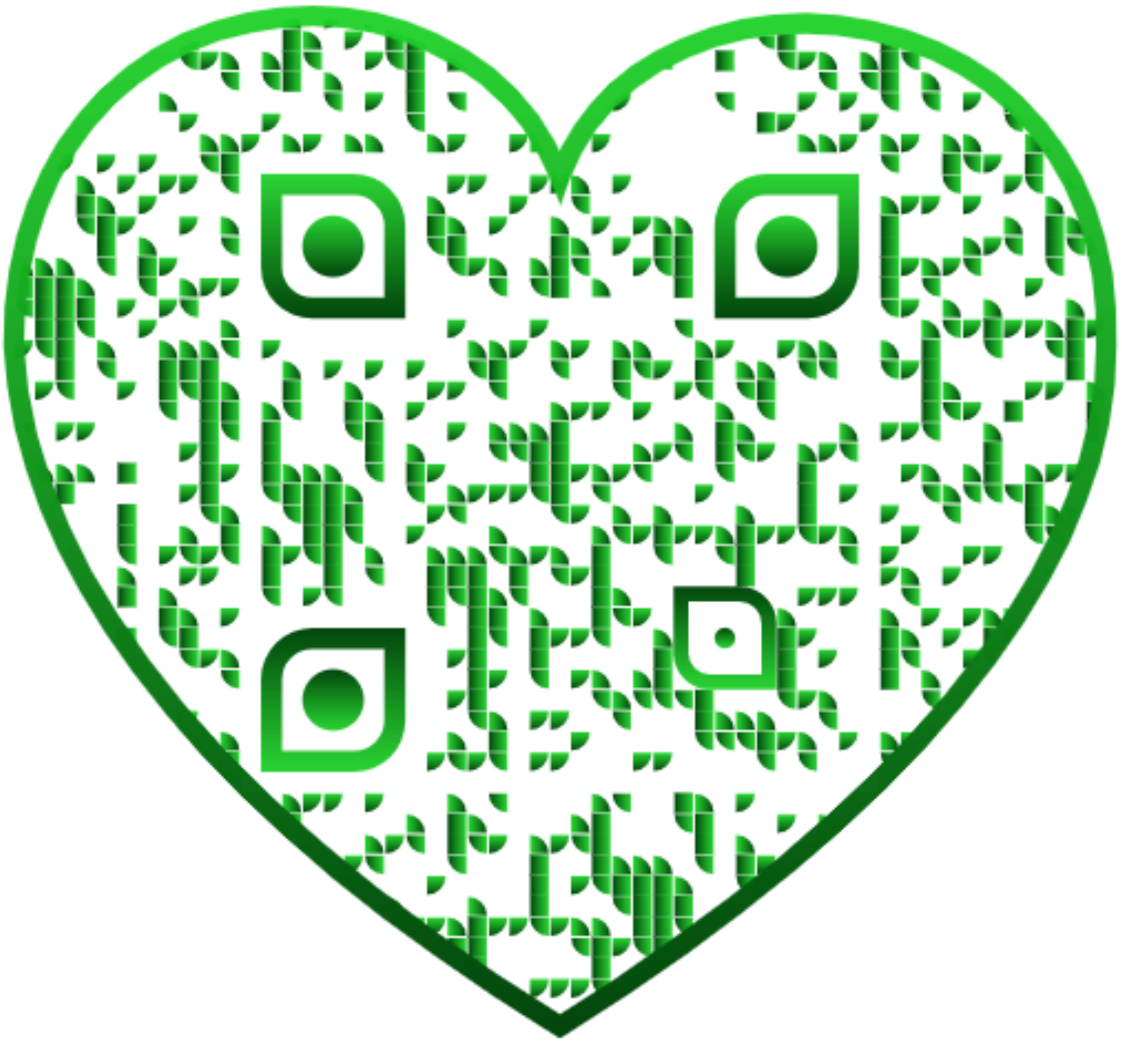
Mikaël Bidault

Programmation Excel avec VBA

Compatible avec toutes les versions d'Excel

EYROLLES



Résumé

MAÎTRISEZ TOUS LES ASPECTS DE LA PROGRAMMATION EXCEL

Qu'il s'agisse de faire face à un besoin immédiat ou de créer des programmes durables, cet ouvrage vous aidera à tirer pleinement profit d'Excel grâce à la programmation VBA.

Vous y apprendrez les principes de la programmation orientée objet, le langage VBA et découvrirez en détail Visual Basic Editor, l'environnement de programmation Excel.

Du simple enregistrement de macros à la conception d'interfaces utilisateur et au débogage de vos programmes, vous trouverez ici toutes les informations nécessaires au développement d'applications VBA pour Excel.

DÉVELOPPEZ UNE APPLICATION EXCEL PROFESSIONNELLE

Vous finirez l'ouvrage par la création d'un programme complet de génération de contrats et de feuilles de paie via une série d'interfaces utilisateur qui vous permettront de récapituler et mettre en œuvre toutes les connaissances acquises lors de votre lecture.

À qui s'adresse cet ouvrage ?

- Aux utilisateurs d'Excel désireux d'améliorer leur productivité
- Aux responsables qui souhaitent créer des solutions sûres et efficaces pour leurs équipes
- Aux personnes qui souhaitent s'initier à la programmation via le tableur de Microsoft

Des compléments web à télécharger

Tous les exemples de programmes du livre sont en téléchargement sur notre site Internet www.editions-eyrolles.com/dl/0067401.

Sommaire

Découvrir la programmation Excel. Notions fondamentales de la programmation orientée objet (POO) • Premières macros • Déplacement et sélection dans une macro Excel • Découvrir Visual Basic Editor • Programmer en Visual Basic. Développer dans Visual Basic Editor • Variables et constantes • Contrôler les programmes VBA • Fonctions Excel et VBA • Manipuler des chaînes de caractères • Déboguer et gérer les erreurs • Intégrer des applications VBA dans l'interface d'Excel • Développer des interfaces utilisateur. Créer des interfaces utilisateur • Exploiter les propriétés des contrôles • Maîtriser le comportement des contrôles • Notions avancées de la programmation Excel. Programmer des événements Excel • Protéger et authentifier des projets VBA • Exemple complet d'application Excel

Biographie auteur

Éditeur et développeur indépendant, **Mikaël Bidault** développe des compléments Word et Excel pour des maisons d'édition et des sites Internet. Il est le créateur de l'articho, un complément VBA pour Word dédié à l'édition print et numérique (www.articho.eu).

DANS LA MÊME COLLECTION

K. NOVAK. – **Débuter avec LINUX.**
N°13793, 2017, 522 pages.

P. MARTIN, J. PAULI, C. PIERRE DE GEYER. – **PHP 7 avancé.**
N°14357, 2016, 732 pages.

L. BLOCH, C. WOLFHUGEL, A. KOKOS, G. BILLOIS, A. SOULLIÉ, T. DEBIZE. – **Sécurité informatique.**
N°11849, 5e édition, 2016, 648 pages.

R. GOETTER. – **CSS 3 Flexbox.**
N°14363, 2016, 152 pages.

W. MCKINNEY. – **Analyse de données en Python.**
N°14109, 2015, 488 pages.

E. BIERNAT, M. LUTZ. – **Data science : fondamentaux et études de cas.**
N°14243, 2015, 312 pages.

B. PHILIBERT. – **Bootstrap 3 : le framework 100 % web design.**
N°14132, 2015, 318 pages.

C. CAMIN. – **Développer avec Symfony2.**
N°14131, 2015, 474 pages.

S. PITTION, B. SIEBMAN. – **Applications mobiles avec Cordova et PhoneGap.**
N°14052, 2015, 184 pages.

H. GIRAUDEL, R. GOETTER. – **CSS 3 : pratique du design web.**
N°14023, 2015, 372 pages.

C. DELANNOY. – **Le guide complet du langage C.**
N°14012, 2014, 844 pages.

K. AYARI. – **Scripting avancé avec Windows PowerShell**.
N°13788, 2013, 358 pages.

W. BORIES, O. MIRIAL, S. PAPP. – **Déploiement et migration Windows 8**.
N°13645, 2013, 480 pages.

W. BORIES, A. LAACHIR, D. THIBLEMONT, P. LAFEIL, F.-X. VITRANT. – **Virtualisation du poste de travail Windows 7 et 8 avec Windows Server 2012**.
N°13644, 2013, 218 pages.

J.-M. DEFRANCE. – **jQuery-Ajax avec PHP**.
N°13720, 4e édition, 2013, 488 pages.

SUR LE MÊME THÈME

D.-J. DAVID. – **VBA pour Excel 2010, 2013 et 2016.**
N°14457, 2016, 324 pages.

N. BARBARY. – **Excel expert**.
N°13692, 2e édition, 2014, 444 pages.

J.-M. LAGODA, F. ROSARD. – **Réaliser des graphiques avec Excel**.
N°56425, 2016, 128 pages.

Retrouvez nos bundles (livres papier + e-book) et livres numériques sur
http://izibook.eyrolles.com

Mikaël Bidault

Programmation Excel avec VBA

EYROLLES

ÉDITIONS EYROLLES
61, bd Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com

Attention : pour lire les exemples de lignes de code, réduisez la police de votre support au maximum.

En application de la loi du 11 mars 1957, il est interdit de reproduire intégralement ou partiellement le présent ouvrage, sur quelque support que ce soit, sans l'autorisation de l'Éditeur ou du Centre Français d'exploitation du droit de copie, 20, rue des Grands Augustins, 75006 Paris.

© Groupe Eyrolles, 2017, ISBN : 978-2-212-67401-9

Table des matières

Introduction

Compléments VBA et compléments Office

VBA, pour quoi faire ?

Des programmes
Une application hôte et des projets
Un langage de programmation
Un environnement de travail
Conventions typographiques

Codes sources des exemples du livre

PREMIÈRE PARTIE
Découvrir la programmation Excel

CHAPITRE 1
Notions fondamentales de la programmation orientée objet (POO)

Comprendre le concept d'objet

Objets et collections d'objets
Application hôte et modèles d'objets
Accéder aux objets
Les propriétés
Les méthodes
Les événements
Les fonctions

Le modèle d'objets d'Excel

CHAPITRE 2
Premières macros

Créer une macro GrasItalique

Afficher l'onglet Développeur

Démarrer l’enregistrement
Enregistrer les commandes de la macro
Exécuter la macro
Structure de la macro
Améliorer la macro

Une autre méthode d’enregistrement
Enregistrement
Structure de la macro

Écrire la macro
Exécution de la macro

Choisir l’accessibilité des macros
Accessibilité globale ou limitée
Classeurs et modèles
Le classeur de macros personnel
Les macros complémentaires
Définir le classeur de stockage lors de l’enregistrement d’une macro
Accéder aux macros d’un classeur spécifique

CHAPITRE 3
Déplacement et sélection dans une macro Excel

Méthodes de sélection dans une feuille Excel
Clavier
Souris
Notion de cellule active
Références relatives et références absolues

Coder les déplacements effectués lors de l’enregistrement d’une macro
Référence absolue aux cellules
Référence relative aux cellules
Référence aux cellules en fonction de leur contenu
Référence aux plages de cellules nommées

CHAPITRE 4
Découvrir Visual Basic Editor

Accéder à Visual Basic Editor

Les outils et les fenêtres de Visual Basic Editor
L’Explorateur de projet
L’Explorateur d’objets

La fenêtre UserForm
La fenêtre Code
La fenêtre Propriétés
Les barres d'outils
Paramétrer Visual Basic Editor

DEUXIÈME PARTIE
Programmer en Visual Basic

CHAPITRE 5
Développer dans Visual Basic Editor
Structure des programmes Visual Basic
Les modules
Les procédures
Les instructions
Les différents types de procédures
Procédures Sub
Procédures Function
Procédures Property
Des projets bien structurés
Ajouter un module
Supprimer un module
Créer une procédure
Écrire l'instruction de déclaration
La boîte de dialogue Ajouter une procédure
La notion de portée
Écriture et mise en forme du code
Déplacer une procédure
Appel et sortie d'une procédure
Appel d'une procédure Sub
Appels de procédures Function et Property
Passage d'arguments
Sortie d'une procédure
Sortie d'un programme
Exécuter du code
Aide à l'écriture de code
Vérification automatique de la syntaxe

Complément automatique des instructions
Info express automatique

CHAPITRE 6
Variables et constantes

Déclarer une variable
Déclaration implicite
Déclaration explicite

Types de données des variables
Chaînes de caractères
Valeurs numériques
Valeurs booléennes
Dates
Type Variant
Variables de matrice
Variables objets
Types de données personnalisés
Constantes
Validation et conversion des types de données

Portée et durée de vie des variables
Portée de niveau procédure
Portée de niveau module privée
Portée de niveau module publique
Variables statiques

Traitement entre applications à l'aide de variables objets

CHAPITRE 7
Contrôler les programmes VBA

Répéter une série d'instructions : les boucles
La boucle While…Wend
La boucle Do…Loop
La boucle For…Next
La boucle For Each…Next

Utiliser des instructions conditionnelles
La structure de contrôle If…Then…Else
La structure de contrôle Select Case

Définir l'instruction suivante avec GoTo

Interagir avec l'utilisateur via des boîtes de dialogue
La fonction InputBox
La méthode InputBox
La fonction MsgBox
Affichage de boîtes de dialogue Excel
Utiliser les opérateurs logiques
Trier des données

CHAPITRE 8
Fonctions Excel et VBA
Utiliser les fonctions Excel dans VBA
Créer des fonctions Excel personnalisées
Intégrer une fonction via l'Explorateur d'objets
Insérer une fonction VBA dans votre code
Insérer une fonction Excel dans votre code
Recommandations pour l'écriture de fonctions Excel
Les limites de la cellule
Principales fonctions VBA

CHAPITRE 9
Manipuler des chaînes de caractères
Modifier des chaînes de caractères
Concaténer des chaînes
Insérer des caractères non accessibles au clavier
Répéter une série de caractères
Supprimer les espaces superflus d'une chaîne
Extraire une partie d'une chaîne
Effectuer des remplacements au sein d'une chaîne
Modifier la casse des chaînes de caractères
Comparer des chaînes de caractères
Rechercher dans les chaînes de caractères
Rechercher une chaîne dans une chaîne
Scinder une chaîne
Rechercher une chaîne dans une variable de matrice

CHAPITRE 10

Déboguer et gérer les erreurs

Les étapes et les outils du débogage

Test du projet
Exécuter pas à pas
La fenêtre Variables locales
Les points d'arrêt
Modifier l'ordre d'exécution des instructions
La fenêtre Exécution
Les espions
La pile des appels

Exemple de débogage

Recherche du bogue
Résolution du bogue

Gestion des erreurs et des exceptions

Exemple de gestion d'erreur

CHAPITRE 11

Intégrer des applications VBA dans l'interface d'Excel

Affecter une macro à un raccourci clavier

Personnaliser le ruban et la barre d'outils Accès rapide

Affecter une macro à un bouton

Affecter une macro à un objet

TROISIÈME PARTIE

Développer des interfaces utilisateur

CHAPITRE 12

Créer des interfaces utilisateur

Les phases de développement de feuilles

Créer une feuille

Les contrôles de la boîte à outils

Outil Sélection
Contrôle Label
Contrôle TextBox
Contrôle ComboBox
Contrôle Frame

Contrôle ListBox
Contrôle CheckBox
Contrôle OptionButton
Contrôle ToggleButton
Contrôle CommandButton
Contrôle TabStrip
Contrôle MultiPage
Contrôle ScrollBar
Contrôle SpinButton
Placer des contrôles sur une feuille
Copier-coller des contrôles
Sélectionner plusieurs contrôles
Supprimer des contrôles
Mise en forme des contrôles
La grille
Aligner les contrôles
Uniformiser la taille des contrôles
Uniformiser l'espace entre les contrôles
Centrer les contrôles
Réorganiser les boutons de commande
Grouper ou séparer des contrôles
Personnaliser la boîte à outils
Ajouter/supprimer un contrôle
Ajouter/supprimer une page
Afficher/masquer une feuille

CHAPITRE 13
Exploiter les propriétés des contrôles
Propriété Name
Apparence
Alignment
BackStyle
Color
BorderStyle
BorderColor
Caption
ControlTipText

- ForeColor
- SpecialEffect
- Style
- Value
- Visible

Comportement

- AutoSize
- AutoTab
- AutoWordSelect
- Cancel
- Default
- Enabled
- EnterKeyBehavior
- HideSelection
- Locked
- MaxLength
- MultiLine
- SelectionMargin
- Style
- TabKeyBehavior
- TextAlign
- TripleState
- WordWrap

Défilement

- ScrollBars
- KeepScrollsVisible
- Delay
- Max et Min
- SmallChange
- LargeChange

Divers

- Accelerator
- GroupName
- HelpContextID
- MouseIcon
- MousePointer
- TabIndex
- TabStop

Tag
Emplacement
Height et Width
Left et Top
StartUpPosition
Image
Picture
PictureAlignment
PicturePosition
PictureSizeMode
PictureTiling
Police
Font

CHAPITRE 14
Maîtriser le comportement des contrôles
Créer des procédures événementielles
Créer une procédure
Les événements
Exemples d'exploitation des contrôles
Label
Contrôle TextBox
ComboBox
ListBox
CheckBox et OptionButton
ScrollBar
SpinButton
Exploiter les informations d'une feuille VBA

QUATRIÈME PARTIE
Notions avancées de la programmation Excel

CHAPITRE 15
Programmer des événements Excel
L'objet Application
Déclaration et instanciation de l'objet Application
Création de procédures événementielles de niveau application

Propriétés de l'objet Application
Méthodes de l'objet Application
L'objet ThisWorkbook
L'objet Worksheet

CHAPITRE 16
Protéger et authentifier des projets VBA
Les virus macros
Se protéger des virus macros
Définir un niveau de sécurité
Les signatures numériques
Sauvegarder des macros
Protéger l'accès aux macros
Verrouiller un projet
Limiter les droits d'exécution d'une macro
Authentifier ses macros
Obtenir une authentification
Authentifier une macro

CHAPITRE 17
Exemple complet d'application Excel
Présentation du projet d'application Excel
Avant de commencer
Identification des informations à recueillir
Définition de la structure du programme
Créer un modèle Excel
Définir et créer des interfaces
Feuille fmContratAuteur
Feuille fmContratConditions
Feuille fmContratDates
Feuille fmContratImpression
Feuille fmContratFin
Écriture des procédures d'édition de documents
Édition des feuilles de paie
Mise à jour du tableau Word

ANNEXE

Mots-clés pour la manipulation de fichiers et de dossiers

Index

Introduction

Visual Basic pour Applications, VBA, est la solution de programmation intégrée aux applications de la suite Office. La connaissance de VBA permet à l'utilisateur d'Excel de tirer pleinement profit du tableur de Microsoft en en développant les capacités et les fonctionnalités pour ses besoins spécifiques. Maîtriser VBA, c'est à coup sûr améliorer sa productivité.

L'intégration dans Excel de Visual Basic pour Applications, un *environnement de développement intégré* complet et professionnel, remonte à sa version 97. Office 2013 et Office 2016 intègrent la version 7.1 de Visual Basic, tandis qu'Office 2010 propose la version 2010 et que XP, 2003 et 2007 fournissent Visual Basic 6.3. Entre ces versions, les différences sont quasi-inexistantes.

Cet ouvrage traite de la programmation des versions 97 à 2016 d'Excel. Sauf exception signalée, les explications et les exemples proposés sont valides pour toutes les versions d'Excel. En effet, de l'une à l'autre, il n'y a pas eu de révolution. Le modèle d'objets s'est affiné et les nouvelles fonctions d'Excel, apparues au cours des différentes versions du logiciel, peuvent également être manipulées *via* la programmation VBA. Cependant, le langage, la gestion des programmes, l'environnement et les outils au service du développeur – bref, tout ce que vous devez savoir pour programmer Excel et que cet ouvrage se propose de vous apprendre – restent inchangés d'une version à l'autre.

Donc, sachez que vous pourrez appliquer les connaissances acquises lors de la lecture de ce livre, aussi bien avec Excel 2003 sous Windows XP qu'avec la version 2016 et un système Windows 10. Mieux, les programmes développés pour Excel 97 fonctionnent avec toutes les versions ultérieures du tableur et, dans la très grande majorité des cas, les programmes développés dans Excel 2016 devraient fonctionner avec les versions antérieures.

Dans cet ouvrage, vous découvrirez les différentes méthodes de création de projets VBA pour Excel, Visual Basic (le langage de programmation proprement dit) et les outils de développement et de gestion intégrés de Visual Basic pour Applications. Votre initiation à la programmation VBA se fera au moyen d'exemples de programmes détaillés et commentés.

Définition

Vous rencontrerez le terme « projet » tout au long de cet ouvrage. C'est ainsi que l'on nomme un ensemble de programmes développés avec Visual Basic pour Applications.

Compléments VBA et compléments Office

Avec Office 2013, Microsoft a introduit un nouveau type de compléments, les compléments Office. Contrairement à ceux développés en VBA, les compléments Office ne sont pas installés sur l'ordinateur de l'utilisateur, mais hébergés sur un serveur distant à partir duquel ils s'exécutent. Ils sont développés à partir de technologies Web, telles que HTML 5, JavaScript, CSS 3, XML et des API REST.

En termes d'expérience utilisateur, les compléments Office s'apparentent à des applications mobiles auxquelles on s'abonne *via* l'Office store et s'exécutent systématiquement dans un panneau qui leur est dédié. Cependant, si vous souhaitez développer des solutions professionnelles dans le cadre d'une entreprise et non dans le but de les commercialiser *via* l'Office store, VBA reste presque toujours la solution la plus simple et la plus souple à mettre en œuvre et à déployer.

VBA, pour quoi faire ?

Excel offre des possibilités très étendues. Pourtant, quelle que soit la puissance de ses fonctions, elles ne peuvent répondre à toutes les situations. La programmation VBA est la solution de personnalisation offerte par Excel, afin d'ajouter des caractéristiques, des fonctions et des commandes qui répondent précisément à vos besoins.

La programmation VBA peut être définie comme la *personnalisation d'un logiciel afin de s'assurer gain de temps, qualité des documents et simplification des tâches complexes ou fastidieuses.* Voici quelques exemples de ce que permettent les programmes VBA :

- Combiner un nombre indéterminé de commandes. Nous sommes souvent amenés à répéter ou à associer certaines commandes plutôt que d'autres et à ignorer certaines fonctionnalités selon l'usage personnel que nous avons d'un logiciel. VBA permet d'associer un nombre illimité de commandes à une seule. Vous pouvez ainsi ouvrir simultanément plusieurs documents Excel stockés dans des dossiers ou sur des serveurs différents, y insérer des

données spécifiques et leur appliquer des mises en forme adaptées, en exécutant une seule commande créée en VBA.

- Ajouter de nouvelles commandes et de nouvelles fonctions à Excel – par exemple, une fonction personnalisée qui calcule les taxes à retenir sur un salaire (ou, mieux, les primes à y ajouter), etc. Vous pouvez, en outre, attacher vos programmes VBA à des raccourcis clavier et à des commandes d'onglets afin d'en améliorer l'accessibilité.

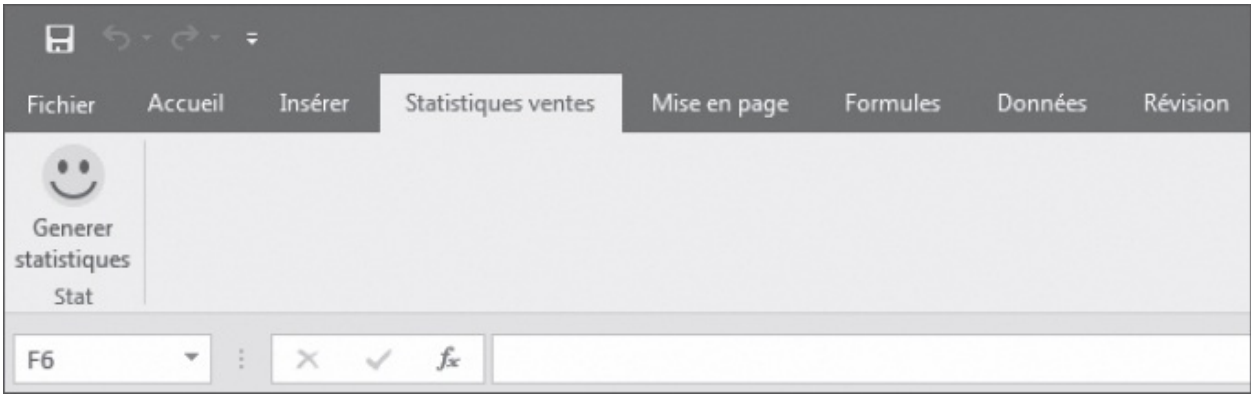


Figure 1 – *Vous pouvez affecter les macros VBA à des commandes sur des onglets personnalisés.*

- Automatiser des actions répétitives. Nous sommes parfois amenés à répéter certaines opérations plusieurs fois sur un même document ou à réitérer des traitements spécifiques. Un programme VBA peut, par exemple, mettre en forme des cellules dans un classeur Excel, effectuer des séries de calculs, etc.
- Modifier et améliorer les commandes d'une application. Les commandes Excel ne sont pas toujours adaptées à nos besoins ou présentent parfois des limitations gênantes. Un programme VBA peut modifier, brider ou compléter les commandes d'une application. Vous pouvez ainsi intégrer dans un tableau le nom de l'utilisateur, le nombre de pages imprimées et l'imprimante utilisée chaque fois qu'une impression est lancée à partir d'Excel.
- Faire interagir les différentes applications Office. Un programme VBA sait exploiter des données issues de fichiers générés par d'autres programmes et interagir avec ceux-ci de façon transparente pour l'utilisateur. Vous pouvez ainsi créer une commande qui envoie automatiquement le classeur Excel ouvert en fichier joint dans un courriel Outlook à des destinataires définis ou qui génère un rapport Word à partir de données Excel et l'imprime.

- Créer des interfaces personnalisées. Les programmes VBA peuvent ramener des tâches complexes à la simple information de champs dans des boîtes de dialogue personnalisées pour l'utilisateur final, simplifiant ainsi considérablement le travail de celui-ci, tout en vous assurant qu'aucun oubli ou fausse manipulation n'aura lieu.

Visual Basic pour Applications permet le développement de solutions adaptées à vos besoins. Les outils que vous apprendrez à manier vous permettront de développer des programmes simples, sans écrire la moindre ligne de code, comme des programmes complets intégrant une interface utilisateur adaptée.

La fonction d'un programme VBA peut être d'automatiser une tâche répétitive. Cependant, vous pouvez aussi créer très vite un petit programme VBA pour faire face à une nécessité immédiate ; par exemple, afin de généraliser un traitement exceptionnel à l'ensemble d'un document.

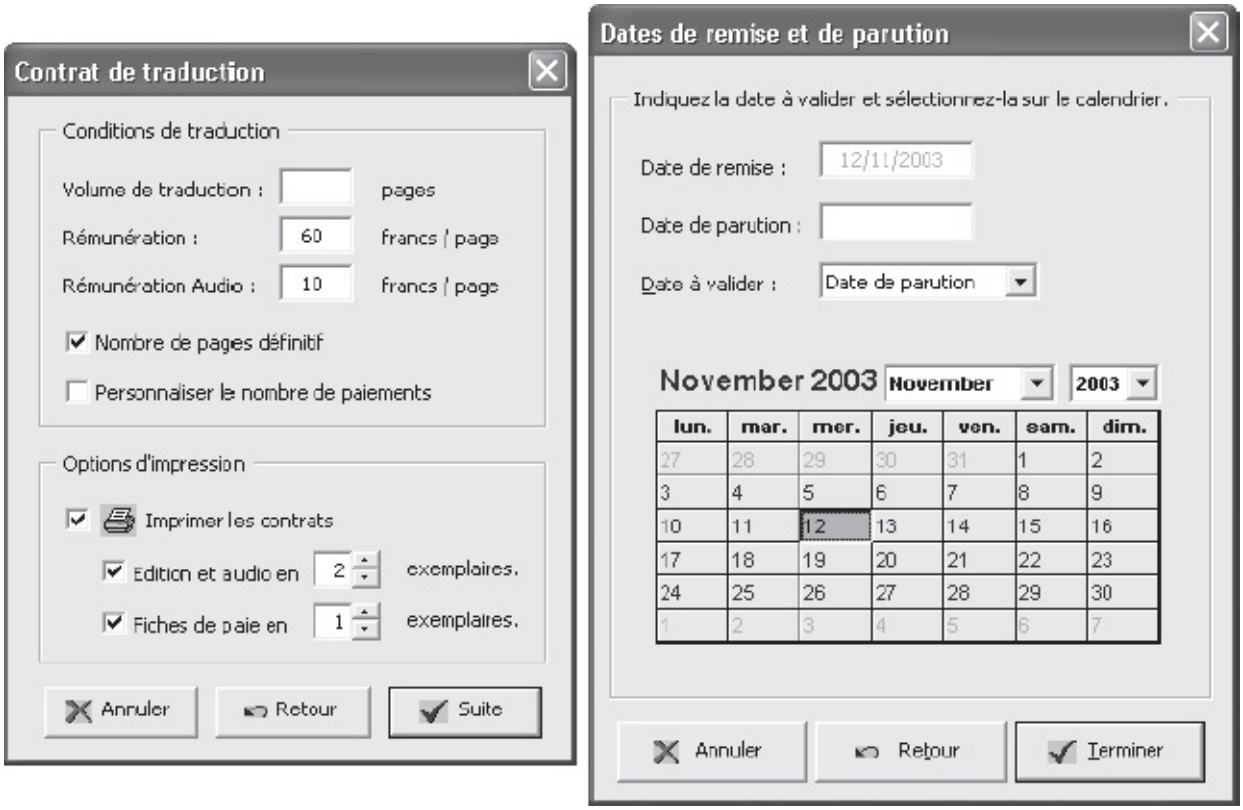


Figure 2 – *Visual Basic pour Applications vous permet de développer des interfaces utilisateur évoluées.*

Des programmes

Les projets *VBA* sont des programmes ou *macros* écrits dans le langage Visual

Basic. Si vous ne possédez aucune expérience préalable en programmation, ne vous inquiétez pas : cet ouvrage aborde le développement de projets VBA à travers l'enregistrement de macros. Lorsque vous l'activez, l'Enregistreur de macro mémorise chacune de vos actions. C'est votre programmeur personnel : vous utilisez simplement les commandes d'Excel et il se charge de traduire les actions exécutées en instructions Visual Basic. Il vous suffit ensuite d'exécuter la macro pour répéter l'ensemble des commandes enregistrées.

Définition

Le terme macro désigne le regroupement d'un ensemble de commandes en une seule. On parle parfois de macrocommande pour désigner un programme qui se résume à l'exécution d'une série de commandes, sans égard pour le contexte. Des macros plus évoluées peuvent répéter des opérations en boucle ou afficher des boîtes de dialogue qui autorisent une interaction avec l'utilisateur. Ces programmes se comporteront différemment en fonction des informations entrées ou de l'état du document sur lequel elles s'exécutent.

Le terme projet est plus large. Il désigne l'ensemble des éléments constituant vos programmes VBA. Il s'agit toujours de macros, mais à celles-ci peuvent s'ajouter des feuilles – qui constituent une interface utilisateur permettant de récolter des informations de tout type –, des modules de classe et autres friandises que vous découvrirez tout au long de cet ouvrage.

L'enregistrement de macros constitue sans aucun doute le meilleur moyen de se familiariser avec la programmation en Visual Basic. Ainsi, sans connaître le langage – les instructions qui le composent et la façon dont elles sont structurées –, vous pouvez créer des programmes VBA et en visualiser ensuite le code.

Une application hôte et des projets

Visual Basic pour Applications est un environnement de développement calqué sur Visual Basic, une solution de développement d'applications Windows. Les structures de contrôle du langage sont les mêmes, et l'environnement proprement dit (Visual Basic Editor) est pour ainsi dire identique à celui de Visual Basic. Cependant, contrairement à Visual Basic, Visual Basic pour Applications est conçu… *pour des applications*. Cela signifie que, tandis que les programmes Visual Basic sont autonomes, les programmes VBA ne peuvent être exécutés qu'à partir d'une application intégrant cet environnement de développement – Excel ou une autre application.

Lorsque vous développez un programme VBA, vous l'attachez à une application. Il s'agit de l'*application hôte* du programme. Plus précisément, vos programmes VBA sont attachés à un document (un fichier ou un modèle

Word, une feuille de calcul Excel, une présentation PowerPoint…) spécifique à l'application hôte. L'ensemble des programmes VBA attachés à un document constitue un projet. Un projet regroupe des macros, mais peut également intégrer des interfaces utilisateur, des déclarations système, etc. Un projet constitue en fait la partie VBA d'un document. Si cet ouvrage ne traite que de la programmation pour Excel, sachez qu'un programme VBA peut être attaché à une autre application. Les concepts et les outils que vous découvrirez au long de cet ouvrage sont valides pour toutes les applications de la suite Office. Pour exécuter une macro VBA, vous devez avoir accès au document auquel elle est attachée. Vous pouvez choisir de rendre certaines macros disponibles à partir de n'importe quel document Excel ou en limiter l'accessibilité à un classeur Excel spécifique. La disponibilité des programmes VBA est abordée au chapitres 2.

Un langage de programmation

Les projets VBA sont développés dans le langage de programmation Visual Basic. Vous découvrirez par la pratique la structure de ce langage et apprendrez rapidement à en discerner les composants et les relations qu'ils entretiennent. Comme nous l'avons dit précédemment, l'enregistrement de macros constitue une excellente initiation à Visual Basic. C'est sous cet angle que nous vous ferons découvrir ce langage.

Visual Basic est un langage de *programmation orientée objet* (POO). Nous présenterons donc les concepts de ce type de programmation. Vous apprendrez ce qu'on appelle un objet, une propriété, une méthode ou un module de classe. Vous verrez comment conjuguer ces éléments pour créer des applications Excel souples et puissantes. Visual Basic pour Applications constitue une bonne approche de la programmation pour le néophyte.

VBA intègre un grand nombre d'instructions, grâce auxquelles vous développerez des macros qui identifient très précisément l'état de l'application et des documents et reproduisent l'exécution de la plupart des commandes disponibles dans l'application hôte.

Vous verrez que certaines instructions sont spécifiques à Excel, par exemple celles qui affectent une formule à une cellule. Vous n'utiliserez probablement qu'un nombre limité de ces instructions, en fonction de votre usage personnel d'Excel ou des besoins de votre entreprise. Cependant, certaines apparaîtront presque toujours dans vos macros. C'est par exemple le cas de la propriété `Range`, qui renvoie un objet Excel tel qu'une cellule ou une plage de cellules.

D’autres instructions sont communes à l’ensemble des applications Office, notamment celles qui règlent le comportement d’une macro : réaliser des opérations en boucle, induire des réactions face à certains paramètres, afficher des boîtes de dialogue simples (figures 3 et 4) ou développer des interfaces utilisateur évoluées (figure 1), etc. Ce sont ces instructions qui constituent véritablement ce qu’il est convenu d’appeler *le langage Visual Basic*. Vous aurez besoin d’y faire appel dès que vous voudrez créer un programme interactif, capable de se comporter différemment selon le contexte. Pour la plupart, ces instructions ne peuvent être générées par enregistrement de macros et doivent donc être éditées manuellement dans Visual Basic Editor.

Figure 3 – *La fonction VBA MsgBox affiche une boîte de dialogue.*

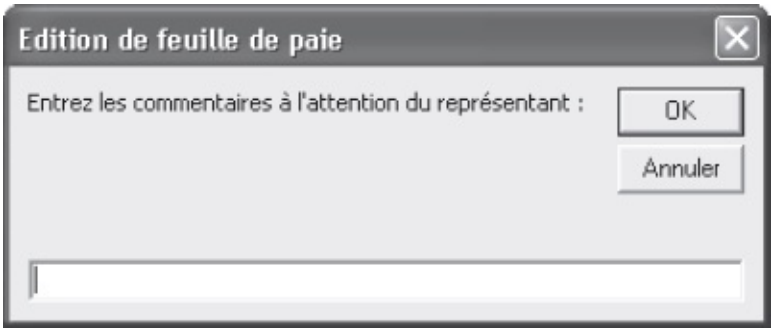


Figure 4 – *Il existe une version VBA et une version Excel de la fonction InputBox.*

Cet ouvrage ne se veut pas un dictionnaire du langage, mais un guide qui vous enseignera le développement de projets VBA de qualité. Vous apprendrez à enregistrer, modifier, exécuter et déboguer des macros, à créer des interfaces utilisateur ainsi qu’à gérer vos projets VBA. Vous découvrirez, à travers les exemples de cet ouvrage, un certain nombre d’instructions spécifiques à la *hiérarchie d’objets* d’Excel, qui vous familiariseront avec la logique de ce langage.

Définition

La hiérarchie d'objets d'une application, encore appelée modèle d'objets, est le rapport qu'entretiennent entre eux les différents objets d'une application. Ce concept ainsi que les notions spécifiques aux langages orientés objet seront développés au chapitres 1, « Notions fondamentales de la programmation orientée objet ».

Ce livre présente et illustre d'exemples commentés l'ensemble des structures de contrôle qui servent à créer très simplement des macros évoluées. Nous vous fournirons les bases du langage Visual Basic. Elles suffisent pour créer une infinité de macros et répondre à vos besoins spécifiques.

Lorsque les principes du développement de projets VBA vous seront acquis et que vous créerez vos propres macros, il vous arrivera sûrement d'avoir besoin d'instructions que vous n'aurez pas rencontrées lors de la lecture de cet ouvrage ; vous pourrez alors utiliser l'Enregistreur de macro ou encore les rechercher dans l'aide de Visual Basic pour Applications ou dans l'Explorateur d'objets – étudié au chapitres 4. Vous verrez que l'aide de VBA fournit une référence complète du langage, facilement accessible et consultable.

Si vous n'avez aucune expérience de programmation, peut-être ce *Visual Basic* vous apparaît-il comme un langage barbare ou inaccessible. Ne vous inquiétez pas : le développement de projets VBA ne requiert ni expérience préalable de la programmation, ni connaissance globale du langage. Contentez-vous, au cours de votre lecture, d'utiliser les fonctions nécessaires aux exercices et que nous vous détaillerons. Cet ouvrage propose un apprentissage progressif et concret : vous développerez vos premiers projets VBA dès les premiers chapitres.

Un environnement de travail

VBA dispose d'un environnement de développement à part entière : Visual Basic Editor.

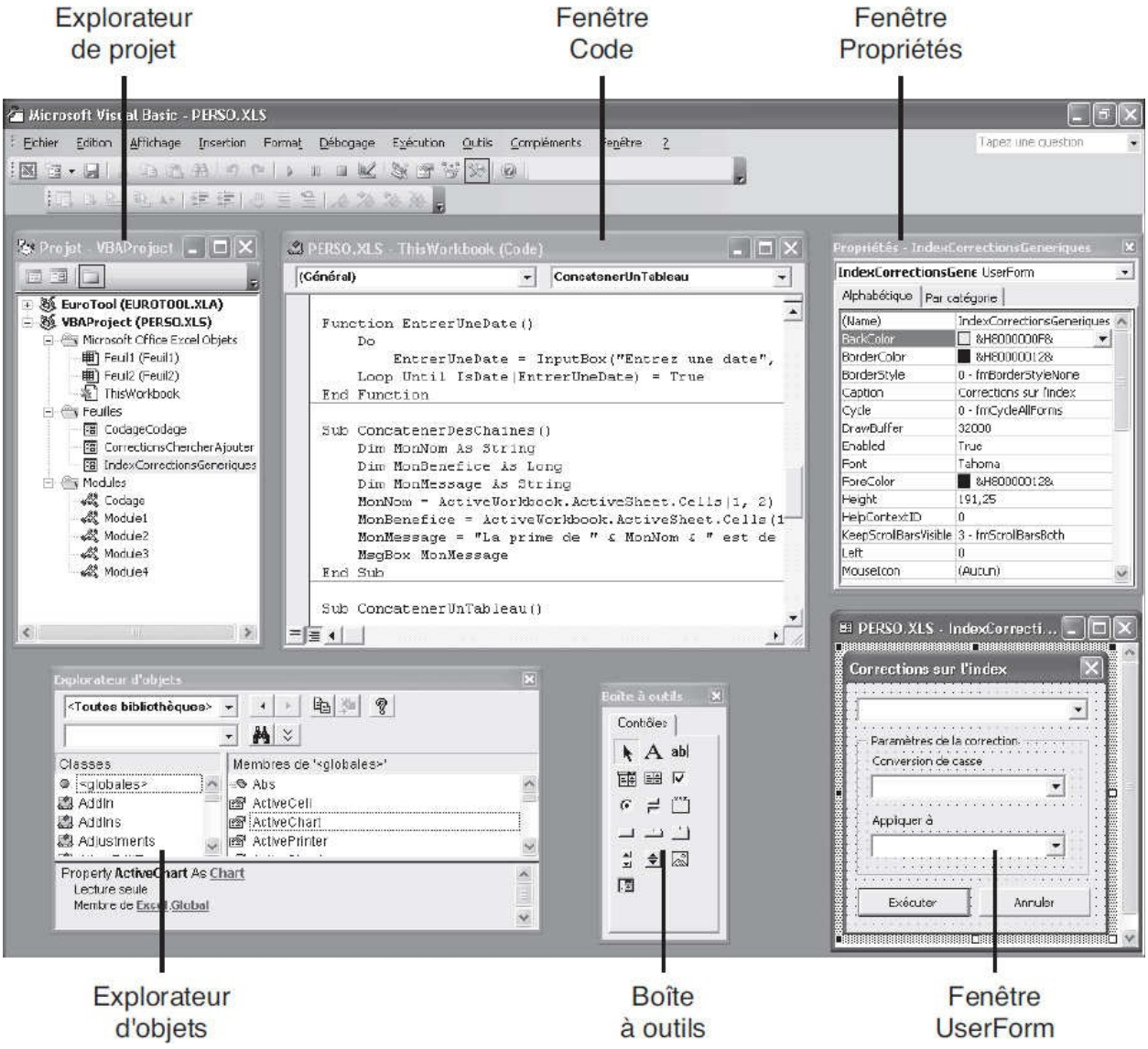


Figure 5 – *Visual Basic Editor est l'environnement de développement de Visual Basic pour Applications.*

Visual Basic Editor est l'*environnement de développement intégré* des applications Office. Il permet de visualiser et de gérer les projets VBA, d'écrire, de modifier et de déboguer les macros existantes, de visualiser comment les commandes propres à une application Office sont traduites en langage Visual Basic et inversement. C'est un outil de débogage de vos projets VBA d'une grande efficacité. Il propose nombre d'outils pour tester les macros et en étudier le comportement. Vous pouvez ainsi exécuter les commandes de la macro pas à pas, en suivre le déroulement, insérer des commentaires dans le texte de la macro, etc. Enfin, cet environnement intègre des outils très intuitifs, dédiés au développement d'interfaces graphiques.

Vous apprendrez dans cet ouvrage à utiliser les nombreux outils de Visual Basic Editor à toutes les phases de développement d'un projet VBA.

Conventions typographiques

Afin de faciliter la lecture, nous avons adopté dans cet ouvrage un certain nombre de conventions typographiques. Lorsqu'un mot apparaît pour la première fois, il est composé en *italique*. Les programmes et les mots-clés du langage Visual Basic apparaissent dans `une police à chasse fixe`. Lorsque, dans un programme, un mot signale une information attendue dans le code, celui-ci apparaît en *`italique`*.

Lorsqu'une ligne de code ne peut être inscrite sur une seule ligne de l'ouvrage, cette flèche (➥) en début de ligne indique que le texte est la suite de la ligne précédente.

Par ailleurs, vous rencontrerez au long de cet ouvrage différents types de notes, matérialisées par des encadrés.

Info

Ces rubriques apportent un complément d'information en rapport avec le sujet traité. Leur lecture n'est pas indispensable, mais elles vous aideront à mieux cerner le sujet.

Définition

Vous trouverez sous ces rubriques la définition de termes techniques spécifiques à la programmation VBA.

Attention

Ces rubriques vous mettent en garde contre les risques inhérents à telle ou telle commande ou manipulation.

Rappel

Il est parfois nécessaire de se rafraîchir la mémoire. Lorsqu'un sujet fait appel à des connaissances acquises plusieurs chapitres auparavant, cette rubrique vous les remémore brièvement.

Astuce

Sous cette rubrique, vous trouverez des trucs pour aller plus vite et travailler plus efficacement.

Conseil

Nous vous faisons ici part de notre expérience, en vous prodiguant des conseils qui vous aideront à développer des projets VBA de qualité.

Codes sources des exemples du livre

Les exemples du livre sont proposés en téléchargement sur le site des éditions Eyrolles. Vous pouvez ainsi tester tous les exemples à partir des fichiers Excel qui les intègrent. Cela vous évitera également de saisir le code dans Visual Basic Editor. Avant de poursuivre la lecture de ce livre, téléchargez les exemples à l'adresse suivante : http://www.editions-eyrolles.com/dl/0067401.

PREMIÈRE PARTIE

Découvrir la programmation Excel

1

Notions fondamentales de la programmation orientée objet (POO)

Visual Basic est un langage de programmation *orienté objet*. En tant que tel, il repose sur des concepts communs à tous les langages de POO. Avant de vous lancer dans la programmation pour Excel, il est important de vous familiariser avec ces concepts et le vocabulaire qui les décrit. Plus concrètement, ce chapitre vous fera découvrir les différents composants de Visual Basic en tant que langage orienté objet et comment ils s'articulent pour créer des programmes VBA puissants.

Vous ne trouverez pas dans ce chapitre de programmes VBA. Il est destiné à vous donner les bases et la terminologie sur lesquelles nous nous appuierons tout au long de cet ouvrage. Alors, patience ! Les connaissances qu'il vous apportera permettront d'appréhender vos premiers programmes dès le chapitres 2.

Comprendre le concept d'objet

Comme pour tous les langages de POO, les *objets* sont le fondement de Visual Basic. Quelle que soit la fonction d'un programme VBA, presque toutes les actions qu'il exécute s'apparentent à la modification d'objets.

Les ouvrages présentant la POO le font presque toujours par analogie avec les objets de la vie réelle. Nous ne dérogerons pas à cette règle. La programmation orientée objet repose en effet sur une structure qui rappelle, par de nombreux points, les objets de la vie courante et les rapports qu'ils entretiennent. Cette analogie rend simples et faciles d'accès des concepts qui, abordés de façon abstraite, vous apparaîtraient probablement obscurs.

Objets et collections d'objets

Dans la vie, un objet peut être tout et n'importe quoi. Ce qui caractérise un objet, c'est son existence physique, ses propriétés spécifiques, son comportement et les actions que l'on peut exécuter sur celui-ci. Une voiture est un objet. Lorsque vous parlez de l'objet `Voiture`, vous pouvez faire référence à un objet abstrait (« Je vais acheter une voiture ») comme à une voiture bien concrète (« Regarde un peu ma belle 504 verte »). Les objets que vous utiliserez dans vos programmes VBA répondent à une même définition.

Dans le premier cas, vous évoquez un objet `Voiture` imprécis et pourtant tout le monde comprend de quoi vous parlez. Il vous suffit de prononcer le mot « voiture » pour que chacun imagine et visualise un véhicule bien spécifique, en fonction de ses goûts, de ses aspirations, de ses souvenirs, etc. Cependant, en tant qu'objet `Voiture`, elle possède un certain nombre de *propriétés* (une carrosserie, des roues, un moteur) et autorise un certain nombre de *méthodes* (démarrer, freiner, tourner) qui permettent d'en maîtriser le comportement.

Ce sont ces *propriétés* et ces *méthodes*, communes à toutes les voitures, qui définissent l'objet `Voiture`. Elles sont sous-entendues, évidentes et essentielles. Il existe donc des milliers de voitures différentes, toutes reconnaissables par un certain nombre de caractéristiques communes définies dans le concept (l'objet) `Voiture`. En POO, cet objet abstrait est appelé la *classe* `Voitures` et est la définition formelle des objets `Voiture` (leurs propriétés et leurs méthodes). Il s'agit du modèle à partir duquel vous pouvez imaginer et créer des milliers de voitures différentes. L'ensemble des véhicules appartenant à la classe `Voitures` (parce qu'ils possèdent les propriétés et les méthodes définies dans cette classe) est appelé la *collection d'objets* `Voitures`.

Info

Une collection porte le nom pluriel des objets qu'elle rassemble.

Ainsi, la collection `WorkBooks` renvoie tous les objets `Workbook`, soit tous les classeurs ouverts, la collection `Sheets`, toutes les feuilles d'un objet `WorkBook`, la propriété `Worksheets`, toutes les feuilles de calcul d'un objet `Workbook`, etc. La section « Le modèle d'objets d'Excel » située en fin de chapitre vous fera découvrir les objets Excel les plus importants.

Définition

Le terme Classe désigne la définition commune d'un ensemble d'objets (qu'est-ce qu'une voiture ?), tandis qu'une Collection désigne l'ensemble des objets appartenant à une classe (toutes les voitures en circulation).

Lorsque vous parlez d'acheter la Peugeot 504 verte de vos rêves, vous évoquez une voiture concrète, bien spécifique. Vous créez une *instance* – on parle aussi d'une *occurrence* – de l'objet `Voiture`. Elle possède toutes les propriétés de la classe `Voitures`, mais ces propriétés sont attachées à des valeurs précises. La carrosserie est verte, la vitesse maximale est de *x* km/h, etc. Vous pouvez maîtriser le comportement de votre voiture à l'aide des méthodes définies dans la classe `Voitures` (`Accélérer`, `Freiner`), mais l'effet précis de ces méthodes est étroitement lié aux propriétés de votre véhicule. La puissance du moteur ne permet pas d'atteindre 200 km/h (mais vous pouvez décapoter !) ; les freins ne sont pas équipés du système ABS, il faut donc telle distance pour freiner, etc.

Un programme VBA peut ainsi créer une feuille de calcul Excel en appliquant la méthode `Add` (ajouter) à la collection `WorkBooks` et déterminer les propriétés de ce classeur (son nom, ses options de protection, le nombre des feuilles qui le composent, etc.)

Info

Lorsque vous créez une instance, cet objet possède toutes les propriétés et méthodes définies dans la classe. Ce principe essentiel de la programmation orientée objet est appelé instanciation.

Le grand intérêt de la programmation orientée objet, c'est qu'il n'est pas indispensable de savoir comment fonctionne un objet pour l'utiliser. Lorsque vous achetez une voiture, vous n'avez pas besoin de savoir comment la carrosserie et le moteur ont été fabriqués, ni comment les différents composants sont assemblés ; vous vous contentez de choisir un modèle, une couleur, etc. Il vous suffit de connaître les méthodes propres à la classe `Voitures` pour l'utiliser. Avec VBA, lorsque vous créez une instance d'un objet, vous en définissez les propriétés sans vous préoccuper de la façon dont celles-ci seront appliquées. Il en va de même pour les méthodes que vous utilisez pour maîtriser le comportement d'un objet. Lorsque vous tournez la clé de contact, le moteur de la voiture démarre, sans que vous ayez à vous soucier du détail des événements et des technologies mises en œuvre.

VBA permet, par exemple, de créer des interfaces graphiques pour vos programmes, en déposant simplement les objets dont vous avez besoin (cases à cocher, zones de texte, boutons de commandes), sur une feuille. Ces objets ont des comportements spécifiques que votre programme exploitera, sans que vous ayez besoin de vous soucier de leur mécanisme interne.

Application hôte et modèles d’objets

Lorsque vous développerez des programmes VBA, vous agirez sur des objets qui varieront en fonction des actions que vous souhaitez que votre programme exécute. Vous définirez et associerez ces objets de façon à créer une application complète. Là encore, l’analogie avec les objets de la vie courante est révélatrice. Les objets que nous utilisons sont généralement ordonnés selon leurs fonctions. Lorsque vous souhaitez vous laver, vous vous dirigez vers la salle de bains ; il s’agit du lieu consacré à la toilette. Vous y trouvez un certain nombre d’objets tels que savon, gant de toilette, dentifrice, brosse à dents, etc. Vous utilisez le savon avec le gant de toilette, le dentifrice avec la brosse à dents, et vous pouvez faire une toilette complète.

Si vous souhaitez manger, c’est dans la cuisine que vous vous orienterez. Vous y trouverez quelques objets disponibles dans la salle de bains (savon, robinet, placard). Vous ne devriez cependant pas y trouver de brosse à dents, ni aucun des objets spécifiques à la toilette. En revanche, vous pourrez utiliser le four, ouvrir le réfrigérateur et utiliser tous les objets spécifiques de la cuisine.

Les applications du Pack Office sont comparables aux pièces de votre maison. Lorsque vous choisissez de développer un projet VBA, vous choisissez une *application hôte*. Il s’agit de l’application Office qui contient les objets sur lesquels vous souhaitez agir. C’est dans cette dernière que vous développerez vos programmes, et c’est uniquement à partir de cette application qu’ils pourront être exécutés. Si vous souhaitez travailler sur des textes, vous choisirez d’entrer dans Word. Pour faire des calculs, vous savez que c’est dans Excel que vous trouverez les objets dont vous avez besoin. Access sert au développement et au maniement des bases de données, et PowerPoint à la création de présentations.

Cependant, à l’image des pièces de votre maison, les applications Office ne sont pas hermétiques. Vous pouvez parfaitement vous préparer un plateau repas dans la cuisine et choisir de manger au lit. De façon semblable, des projets VBA évolués sont capables d’utiliser des objets de différentes applications Office. Un programme développé dans Excel peut utiliser des données stockées dans une base de données Access ou des objets Word pour imprimer un courrier qui accompagnera une facture, et envoyer un message Outlook de confirmation.

Vous devez choisir une application hôte pour votre projet. Deux critères doivent la déterminer :

- Votre programme sera plus performant et plus simple à développer si l'application hôte est celle dans laquelle s'exécute l'essentiel des instructions du programme.
- La présence du programme dans l'application hôte doit être logique, et l'utilisateur final doit y accéder facilement puisque le programme ne pourra être exécuté qu'à partir de celle-ci.

Info

Tous les projets développés dans cet ouvrage seront hébergés dans Excel. Pour accéder aux objets d'une application autre que l'hôte, vous utiliserez la technologie Automation. L'accès aux objets d'une autre application est traité au chapitres 6.

L'application est donc la pièce dans laquelle votre programme s'exécutera. Elle est composée d'un certain nombre d'objets – constituant une *bibliothèque* – dont les rapports sont précisément définis. Les objets d'une application et les rapports qu'ils entretiennent sont représentés sous la forme d'un organigramme. Tout en haut de l'organigramme se trouve l'application (la pièce dans laquelle sont rangés tous les objets). Viennent ensuite les classes d'objets de premier niveau de l'application, auxquelles sont liés d'autres objets ou classes, et ainsi de suite. On appelle cette structure le *modèle d'objets* ou la *hiérarchie de classes* de l'application. La figure 1-1 représente ce qui pourrait être un modèle d'objets sommaire de l'application Salle de bains.

Info

Pour la plupart, les éléments d'Excel peuvent être manipulés dans Visual Basic pour Applications en tant qu'objets. Un classeur, une feuille de ce classeur, une cellule ou une boîte de dialogue Rechercher sont autant d'objets manipulables dans un programme Visual Basic.

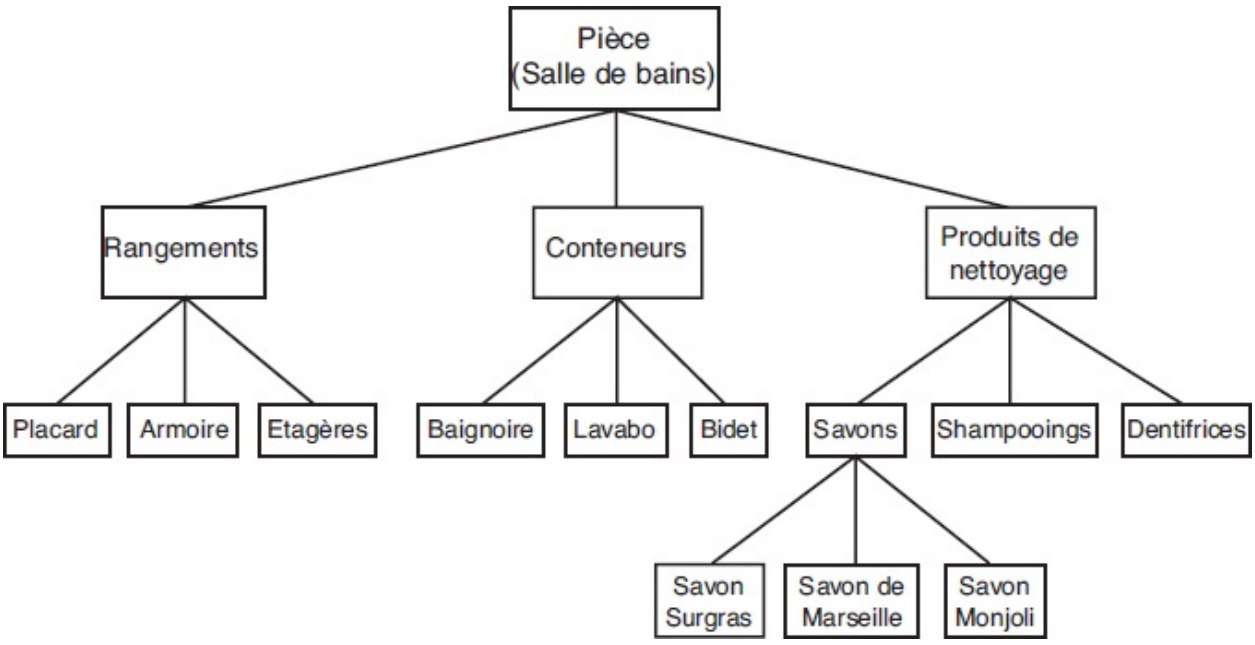


Figure 1-1 – *L'ensemble des objets d'une application est structuré selon un modèle d'objets qui en définit les rapports et la hiérarchie.*

Au sommet du modèle se trouve la pièce – l'application. Tous les objets auxquels vous pouvez accéder y sont contenus. Si l'on établit un modèle d'objets pour l'ensemble des pièces de la maison, on retrouvera toujours l'objet `Pièce` au sommet du modèle. De la même façon, au sommet des modèles d'objets des applications Office, se trouve l'objet `Application`.

Viennent ensuite les classes situées immédiatement sous l'objet `Pièce`. Plus on progresse dans le modèle, plus les objets sont précis et donc spécifiques de la pièce ou de l'application. Dans Excel par exemple, sous l'objet `Application` se trouve la collection (ou classe) `Workbooks` qui englobe tous les objets `Workbook`, c'est-à-dire tous les classeurs Excel ouverts. Sous l'objet `Workbook` se trouve la classe `Worksheets`, qui englobe tous les objets `Worksheet` (toutes les feuilles de calcul) de l'objet `Workbook` désigné.

Astuce

Pour accéder à l'aide en ligne des objets Excel, affichez l'Aide de VBA à partir du menu « ? », puis sélectionnez la commande Référence VBA d'Excel. Vous apprendrez à accéder à Visual Basic Editor au prochain chapitre.

Notez que l'appartenance des objets à des branches distinctes du modèle ne signifie pas qu'ils ne peuvent pas interagir. L'objet `Savon de Marseille` peut se trouver sur l'étagère et vous pouvez utiliser la méthode `Déplacer` pour le mettre

dans l’objet `Baignoire`, comme dans l’objet `Lavabo`.

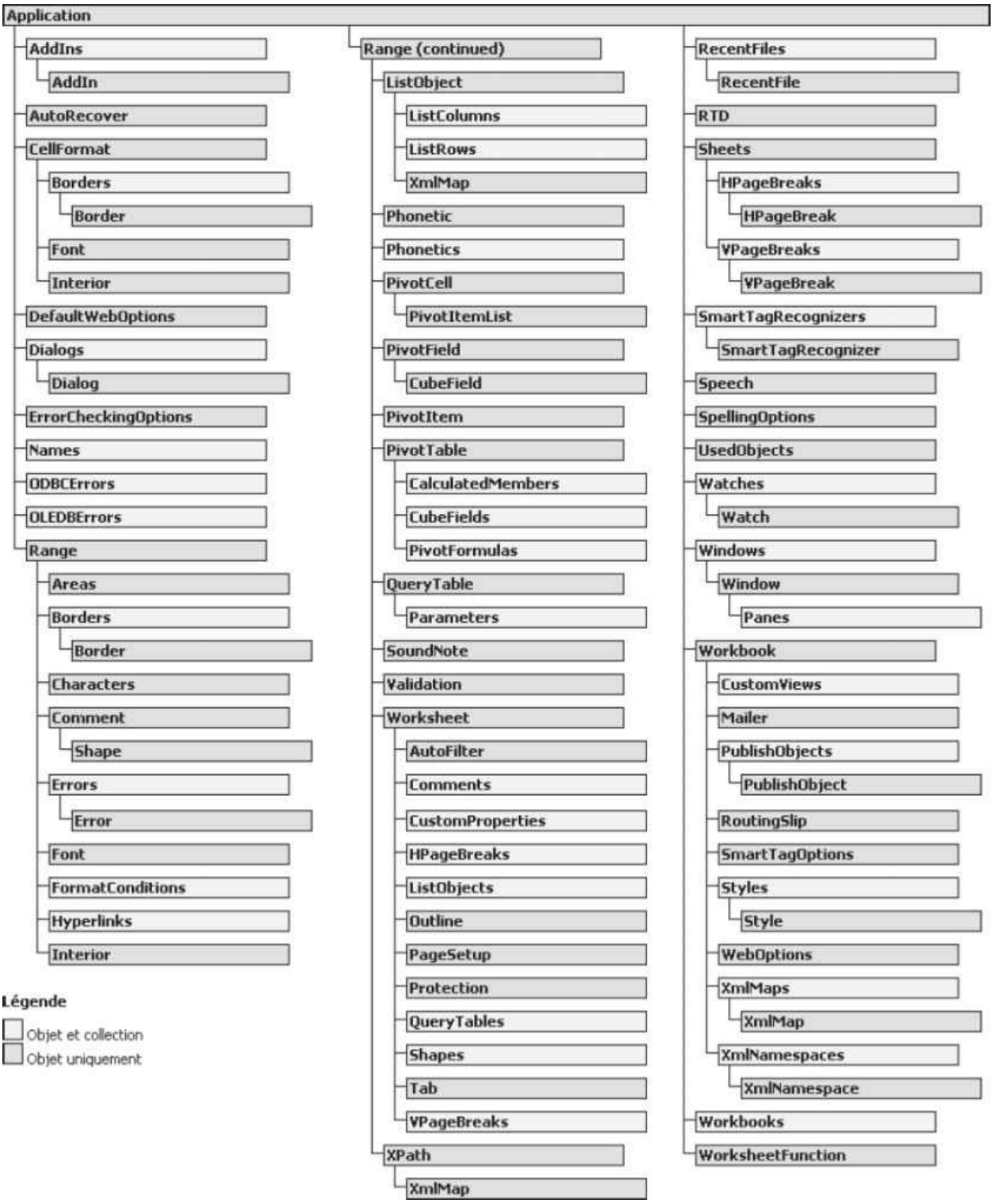


Figure 1-2 – *Le modèle d'objets d'Excel.*

Un objet peut en englober d'autres ; il est alors qualifié de *conteneur*. C'est le cas de l'objet `Application`, mais c'est aussi vrai pour beaucoup d'autres objets du

modèle d'Excel. Par exemple, un `Workbook` contient des `Worksheet` (feuilles de calcul), contenant eux-mêmes des `Range` (cellules et plages de cellules).

Accéder aux objets

Le modèle détermine le chemin à emprunter pour accéder à un objet. Pour vous laver les dents, vous devez d'abord accéder à votre brosse à dents. Même si le processus est inconscient, vous identifiez l'objet `Brosse à dents` par son emplacement : il est situé dans la salle de bains, parmi les objets et produits de toilette. De la même façon, en Visual Basic, vous devez identifier un objet avant de pouvoir agir dessus (appliquer l'une de ses méthodes ou modifier la valeur de l'une de ses propriétés). Lorsque vous souhaitez vous laver les dents, vous pensez et suivez inconsciemment les étapes suivantes :

- aller à la salle de bains ;
- se diriger vers les produits de toilette ;
- choisir parmi ceux-ci la brosse à dents et le dentifrice et s'en saisir.

Pour accéder à un objet Excel, vous opérerez selon le même mode, c'est-à-dire en partant du haut de la hiérarchie et en progressant dans celle-ci jusqu'à atteindre l'objet voulu.

Le point est utilisé comme séparateur entre les différentes collections et objets que l'on rencontre avant d'atteindre l'objet voulu. La référence à un objet précis d'une collection se fait selon la syntaxe suivante :

```
Nom_Collection("Nom_Objet")
```

Le code VBA permettant d'accéder à l'objet `Dentifrice` serait :

```
Piece.ProduitsHygiene("Dentifrice").Prendre
```

La première partie du code permet d'accéder à l'objet `Dentifrice` ; l'expression identifiant un objet est appelée *référentiel d'objet*. La méthode `Prendre` est ensuite appliquée à cet objet afin de s'en saisir.

Le code Visual Basic activant la feuille de classeur Excel, nommée MaFeuille et située dans le classeur MonClasseur.xlsm (à condition que celui-ci soit ouvert), serait :

```
Application.Workbooks("MonClasseur.xlsm").Sheets("MaFeuille").Activate
```

On accède à l'objet `Workbook` MonClasseur de la collection `Workbooks` (tous les classeurs ouverts), puis à la feuille nommée MaFeuille de la collection `Sheets` (toutes les feuilles de l'objet MonClasseur). Une fois le chemin d'accès à

l'objet indiqué, on lui applique la méthode `Activate`.

Info

Outre son nom, chaque objet est identifié par une valeur d'indice représentant sa position dans la collection. Cette valeur peut être utilisée pour renvoyer un objet précis selon la syntaxe suivante :

```
Nom_Collection(IndexObjet)
```

où *`IndexObjet`* représente la position de l'objet dans la collection. L'instruction suivante :

```
Workbooks(2).Activate
```

active le classeur Excel apparaissant en deuxième position dans la liste des fenêtres du bouton Changer de fenêtre de l'onglet Affichage.

Poursuivons l'analogie. Si vous vous trouvez déjà dans la salle de bains au moment où vous décidez de vous laver les dents, vous n'avez pas besoin d'y accéder. Si vous regardez déjà parmi les produits de toilette, il est inutile d'y faire référence.

De façon semblable, dans le code VBA, les objets de *niveau hiérarchique* supérieur à celui de l'objet que vous souhaitez atteindre peuvent parfois être ignorés. C'est toujours le cas pour l'objet `Application`. En effet, votre projet VBA étant stocké et donc exécuté à partir d'une application hôte, il est inutile de rappeler que vous êtes dans cette application.

L'expression :

```
Workbooks("MonClasseur.xlsm").Sheets("MaFeuille").Activate
```

suffit donc à activer la feuille intitulée MaFeuille du classeur MonClasseur.xlsm.

Selon le même principe, en cas d'absence de référentiel d'objets, la collection `Sheets` concerne le classeur actif. Si MonClasseur est le classeur actif, on peut donc se dispenser de toute référence à cet objet. On obtient alors l'instruction suivante :

```
Sheets("MaFeuille").Activate
```

Info

Une petite finesse sémantique : les objets à proprement parler n'apparaissent jamais dans le code. Pour faire référence à un objet, on utilise une propriété qui appelle ou renvoie l'objet voulu. Dans les exemples précédents, `Workbooks` est une propriété de l'objet `Application`, qui renvoie tous les classeurs ouverts (la classe `Workbooks`). `Sheets` est une propriété de l'objet `Workbook`, qui renvoie toutes les feuilles de classeur (la classe `Sheets`) de cet objet.

Les propriétés

Revenons à l'analogie avec l'automobile et prenons la classe `Voitures`. Toutes les propriétés des objets `Voitures` y sont définies. Les objets ou classes situés immédiatement sous `Voitures` dans le modèle appartiennent à la collection d'objets `Voitures`. En tant que tels, ils héritent de toutes les propriétés définies dans la classe `Voitures`.

Les propriétés peuvent être un attribut de l'objet ou un aspect de son comportement. Par exemple, les propriétés d'une voiture sont, notamment, sa marque, son modèle, l'état des pneus, l'activation ou non du moteur, etc. Les propriétés d'un document Word sont son modèle, son nom, sa taille, etc.

Les propriétés prennent des valeurs spécifiques qui distinguent les différents objets de la collection. La propriété `Couleur` d'un objet `Voiture` peut prendre la valeur Vert, tandis que la même propriété d'un objet de la collection est attachée à la valeur Bleu.

Lorsque vous développerez des programmes VBA, vous exploiterez les propriétés d'un objet de deux façons :

- En modifiant les valeurs attachées aux propriétés de l'objet. Les propriétés dont les valeurs peuvent être changées sont dites en lecture-écriture.

  Certaines propriétés ne sont pas modifiables et sont dites en lecture seule. Vous pouvez, par exemple, modifier la propriété `Etat_du_moteur` (allumé ou éteint) d'un objet `Voiture`, mais pas sa propriété `Marque`. Il est possible de changer le nombre de feuilles qui composent un classeur, mais pas sa date de création.
- En interrogeant les valeurs attachées aux propriétés d'un objet. Les valeurs des propriétés peuvent être lues afin de connaître les spécificités de l'objet et d'orienter le comportement du programme. Par exemple, en supposant que le litre d'essence est à 1,50 euros, si la valeur BMW est affectée à la propriété `Marque` d'un objet `Voiture` et la valeur 40 (litres) affectée à sa propriété `Contenu_Réservoir`, vous ferez un plein à 60 euros. Si les valeurs Citroën et 2CV sont respectivement affectées aux propriétés `Marque` et `Modèle` et si la propriété `Contenu_Réservoir` a une valeur égale à 20, vous ne ferez qu'un plein à 30 euros.

Types de valeurs des propriétés

Les valeurs affectées aux propriétés d'un objet peuvent être de quatre types :

- chaîne de caractères ;
- valeur numérique ;

- valeur booléenne ;
- constante.

Chaînes de caractères

Une chaîne de caractères est une suite de caractères contigus – lettres, chiffres, espaces ou signes de ponctuation. Ces données sont aussi qualifiées de type *Chaîne* ou *String*. Une chaîne peut contenir jusqu'à environ deux milliards de caractères. En Visual Basic, les chaînes sont placées entre guillemets :

- "Paul" ;
- "1254" ;
- "Je suis une chaîne de caractères composée de 59 caractères".

Les chaînes sont interprétées en tant que caractères, et non en tant que valeur numérique. Autrement dit, la chaîne "1254" est interprétée comme la combinaison des caractères 1, 2, 5 et 4.

La propriété `Modèle` d'un objet `Voiture` est toujours une chaîne de caractères. Celle-ci ne peut être composée que de chiffres – par exemple "2000" – sans que vous puissiez pour autant diviser cette valeur par un nombre quelconque.

Valeurs numériques

Une valeur numérique est une suite de chiffres. Elle peut être un nombre entier ou décimal, positif ou négatif :

- 0 ;
- 1 548 972 ;
- − 1 245,4542 ;
- 100E4.

Info

Le caractère E dans une variable numérique signifie « exposant » et représente une puissance de 10. Ainsi, la valeur numérique 100E4 est égale à 100×10^4.

Les valeurs numériques sont interprétées comme des chiffres. Il peut s'agir de valeurs comme d'expressions conjuguant valeurs numériques et opérateurs arithmétiques (* / – +). Par exemple, les propriétés `Contenu_Réservoir` et `Consommation` d'un objet `Voiture` sont des valeurs numériques. Leur combinaison détermine combien de kilomètres peuvent être parcourus avant la panne sèche,

selon l'expression arithmétique suivante :

```
Kilomètres_Avant_Panne_Sèche = Contenu_Réservoir / Consommation
```

Info

Notez qu'une expression arithmétique peut être composée de nombres (100 / 25), de variables auxquelles sont affectées des valeurs numériques (nombre1 + nombre2), ou d'une combinaison des deux (nombre1 - 25). Les variables sont étudiées au chapitres 6.

Les valeurs numériques pouvant être affectées à une propriété varient selon les propriétés et les objets. Dans Excel par exemple, la taille d'une police doit être comprise entre 1 et 409. Par conséquent, la valeur que peut prendre la propriété `Size` (taille) d'un objet `Font` (police) d'Excel doit aussi être comprise entre ces deux valeurs. Dans le cas de l'objet `Voiture`, la propriété `Contenu_Réservoir` doit toujours être supérieure à 0, la valeur maximale dépendant d'autres spécificités de l'objet.

Valeurs booléennes

Certaines propriétés ne peuvent prendre que deux états : elles sont vérifiées ou elles ne le sont pas. Elles sont attachées à une valeur de type `Boolean`, ou valeur booléenne, qui vaut `True` ou `False`.

La propriété `Moteur_Allumé` d'un objet `Voiture` est une valeur booléenne : `True` si le moteur de l'objet `Voiture` est allumé, `False` dans le cas contraire.

Comme vous le verrez au chapitres 15, un classeur Excel gère une vingtaine de propriétés qui représentent ses options et son état à un moment donné. Nombre de ces propriétés acceptent une valeur de type `Boolean`. C'est par exemple le cas de `Saved`, qui renvoie `True` si aucune modification n'a été apportée au document depuis son dernier enregistrement, ou `False` dans le cas contraire.

Info

En Visual Basic, la valeur `True` peut être remplacée par -1 et la valeur `False` par 0. Cette pratique est cependant déconseillée, puisqu'elle rend la lecture du code moins aisée.

Constantes

Les constantes sont des valeurs intégrées de VBA qui conservent toujours la même valeur. Lorsqu'une propriété accepte un nombre déterminé d'états, les valeurs représentant ces derniers sont souvent des constantes et se présentent sous la forme d'une suite de lettres. Les constantes sont représentées sous

forme de chaînes de caractères, mais correspondent en réalité à des valeurs numériques.

Les constantes intégrées désignent l'état de propriétés pour un objet spécifique. Chacune des applications Office possède ses propres constantes (puisqu'elle possède ses propres objets). Cependant, certaines propriétés étant communes, les constantes associées se retrouvent aussi dans toutes les applications Office. Une constante intégrée de VBA commence par deux lettres en minuscules indiquant l'application à laquelle elle appartient. Le tableau suivant reprend les préfixes des constantes VBA les plus courantes pour Microsoft Office.

vb	Visual Basic
wd	Word
xl	Excel
pp	PowerPoint
ac	Access
ol	Outlook
fm	Feuilles Visual Basic

Lorsqu'une propriété accepte des constantes pour valeurs, leur nombre est déterminé et correspond aux différents états que peut prendre la propriété. Par exemple, les clignotants d'une voiture peuvent accepter quatre états différents : désactivés, activés à droite, activés à gauche, position Warning (les clignotants droite et gauche activés). La propriété `Etat` d'un objet `Clignotant` pourrait donc accepter l'une des quatre constantes `Clignotant` suivantes :

```
ClignotantAucun
ClignotantDroite
ClignotantGauche
ClignotantWarning
```

Excel intègre de nombreuses constantes. Lorsqu'une commande Excel exige de l'utilisateur la sélection d'une option parmi plusieurs possibles, ces options sont généralement représentées sous forme de constantes en langage VBA. Par exemple, lorsque vous insérez une cellule dans une feuille de classeur (Insertion > Cellules), vous devez choisir entre les options Décaler les cellules vers la droite et Décaler les cellules vers le bas. L'instruction VBA correspondante sera :

```
Selection.Insert(Shift)
```

où l'argument *`Shift`* est une des constantes `XlInsertShiftDirection` spécifiant à la méthode `Insert` la façon dont la cellule sera insérée. Il peut s'agir de la constante `xlShiftToRight` (les cellules seront décalées vers la droite) ou de la

constante `xlShiftDown` (les cellules seront décalées vers le bas).

Les constantes sont la représentation textuelle de valeurs numériques. Chacune des constantes *`Clignotant`* correspond à une valeur numérique. La propriété `ClignotantWarning` correspondrait par exemple à la valeur numérique 3. Vous pouvez indifféremment utiliser les constantes VBA ou les valeurs numériques auxquelles elles correspondent. Il est cependant conseillé d'utiliser les constantes, afin de faciliter la lecture du code.

Accéder aux propriétés

Pour modifier une propriété d'un objet, on utilise la syntaxe suivante :

```
Expression.Propriété = valeur
```

où *`Expression`* est une expression renvoyant un objet – un référentiel d'objet – tel que cela a été décrit dans la section précédente. *`Propriété`* est le nom de la propriété que l'on souhaite modifier (toujours séparée de l'objet auquel elle se réfère par un point) et *`valeur`* est la valeur que vous souhaitez lui affecter.

Le type de la valeur (chaîne, nombre, constante ou booléen) doit être adapté à la propriété. Si tel n'est pas le cas, le programme génère une erreur. Par exemple, la propriété `Contenu_Réservoir` d'un objet `Voiture` n'accepte qu'une valeur numérique ; vous ne pouvez pas lui affecter une chaîne de caractères.

Le tableau 1-1 illustre différentes possibilités de modifier l'objet `Voiture` MaVoiture.

Tableau 1-1. Pour modifier un objet, il suffit d'en changer les propriétés

Syntaxe	Type de la valeur affectée	Conséquence pour l'objet `Voiture`
`Voitures("MaVoiture").Immatriculation = "BS606XH"`	Chaîne de caractères	Une nouvelle immatriculation
`Voitures("MaVoiture").Moteur_Allume = True`	Valeur booléenne	Le moteur est allumé.
`Voitures("MaVoiture").Contenu_Réservoir = 50`	Valeur numérique	Le réservoir contient 50 litres.
`Voitures("MaVoiture").Clignotant.Etat = ClignotantWarning`	Constante	Le clignotant est en position Warning.

Pour mémoriser la valeur d'une propriété d'un objet donné, on la stocke généralement dans une variable, selon la syntaxe suivante :

```
variable = Expression.Propriété
```

L'instruction suivante passe la fenêtre active en mode d'affichage Aperçu des sauts de page, en définissant sa propriété `View` à `xlPageBreakPreview`.

```
ActiveWindow.View = xlPageBreakPreview
```

L'instruction suivante stocke dans la variable `TypeAffichage` la valeur représentant le type d'affichage en cours :

```
TypeAffichage = ActiveWindow.View
```

Les méthodes

Les méthodes représentent les actions qu'un objet peut exécuter. Tandis que les propriétés définissent un état, les méthodes déterminent un comportement et elles dépendent étroitement de l'objet. Les objets de la classe `Voitures` disposent de méthodes telles que `Tourner`, `Freiner`, `Accélérer`, etc.

Cependant, certaines méthodes peuvent être communes à des objets différents, même si elles ont des conséquences différentes. Par exemple, la méthode `Ouvrir` peut s'appliquer aux objets `Porte`, `Coffre` ou `Cendrier` d'une voiture, comme à un objet `Porte` ou `Robinet` d'une maison. Certaines méthodes se retrouvent dans toutes les applications Office. C'est le cas pour toutes les commandes communes aux applications. Par exemple, `Open` (ouvrir) et `Close` (fermer) s'appliquent aussi bien à un classeur Excel qu'à un document Word, un formulaire Access ou encore une présentation PowerPoint.

Une méthode peut avoir des conséquences sur l'état de certaines propriétés de l'objet auquel elle s'applique, voire sur d'autres objets. Par exemple, si vous appliquez la méthode `Accélérer` à un objet `Voiture`, la valeur affectée à la propriété `Vitesse` de cet objet augmentera.

Si vous modifiez le contenu d'une cellule d'un classeur Excel, la taille de la cellule sera peut-être modifiée en conséquence. Si d'autres cellules sont liées par des formules à la cellule dont vous modifiez la valeur, leurs valeurs seront mises à jour en conséquence. Chaque fois que vous créez un nouveau classeur à l'aide de la méthode `Add`, la valeur de la propriété `Count` de la collection `Workbooks` (le nombre de classeurs ouverts) est incrémentée de 1. Chaque fois que vous fermez le classeur à l'aide de la méthode `Close`, la valeur de la propriété `Count` de la collection `Workbooks` est décrémentée de 1.

En outre, pour exécuter correctement une méthode, il est parfois nécessaire de modifier au préalable les propriétés de l'objet auquel elle s'applique. Par exemple, si vous souhaitez appliquer la méthode `Tourner` à un objet `Voiture`, vous devez auparavant modifier la propriété `Etat_Clignotant` de l'objet `Clignotant` de

cette voiture.

La syntaxe pour appliquer une méthode à un objet est la suivante :

```
Expression.Méthode
```

où *Expression* est une expression renvoyant un objet – un référentiel d'objet – et *Méthode* est le nom de la méthode que l'on souhaite exécuter (toujours séparée de l'objet auquel elle se réfère par un point).

Une méthode peut aussi s'appliquer à une collection d'objets :

```
Collection.Méthode
```

Vous pouvez, par exemple, arrêter tous les objets de la collection `Voitures` :

```
Voitures.Arrêter
```

Pour fermer tous les classeurs ouverts dans une session Excel, vous utiliserez l'instruction suivante :

```
Workbooks.Close
```

Cette syntaxe est aussi utilisée pour créer une occurrence d'un objet dans une collection. La méthode utilisée est alors généralement `Add` – l'équivalent Visual Basic de l'onglet Fichier. Par exemple, pour créer un nouveau classeur Excel, vous ferez appel à `Workbooks.Add`.

Vous définissez ensuite les propriétés de l'objet ainsi créé, comme nous l'avons vu dans la section « Les propriétés » de ce chapitre.

Les événements

Un événement est une action reconnue par un objet et qui déclenche l'exécution d'un programme lorsqu'elle survient. On parle alors de *procédure événementielle*. Un clic de souris ou la frappe d'une touche sont des exemples d'événements pouvant être interprétés par un programme VBA.

Définition

Une procédure événementielle est une procédure attachée à un événement utilisateur tel qu'un clic de souris, la frappe d'une touche, l'activation d'une feuille de calcul, etc. La procédure s'exécute lorsque l'événement auquel elle est attachée est reconnu par l'application.

Les objets de la collection `Voitures` reconnaîtront par exemple l'événement `Choc`, dont la détection entraînera l'ouverture de l'objet `Airbag`, autrement dit l'application de la méthode `Ouvrir` à ce dernier.

Les événements s'utilisent essentiellement avec les contrôles de formulaires que vous développerez et avec les objets. Vous apprendrez à exploiter les événements utilisateur affectant un formulaire aux chapitres 13 et 14. Les feuilles de calcul, les graphiques, les classeurs et l'application Excel gèrent aussi des événements. Vous apprendrez à créer des procédures événementielles pour ces objets au chapitres 15.

Les fonctions

Les fonctions servent à renvoyer une information selon les éléments qui leur sont fournis. Le type de l'information renvoyée varie d'une fonction à l'autre. Il peut s'agir d'une chaîne de caractères, d'une valeur numérique, booléenne, de type `Date`, etc. Visual Basic intègre un certain nombre de fonctions exploitables directement. Par exemple, `Asc` renvoie le code ASCII du caractère sélectionné, tandis que `Int` renvoie la partie entière d'un nombre. Certaines fonctions sont particulièrement utiles. C'est le cas de `MsgBox`, qui affiche une boîte de dialogue contenant des boutons (tels que Oui, Non ou Annuler) et qui renvoie une valeur reflétant le choix de l'utilisateur.

Vous créerez aussi vos propres fonctions pour traiter les valeurs qui leur seront passées et renvoyer une valeur ensuite utilisée par le programme. Dans le cas d'un objet `Voiture`, vous pouvez créer une fonction `Coût_Plein` qui exploitera les propriétés `Contenu_Réservoir` et `Contenance_Réservoir` de l'objet, ainsi qu'une variable représentant le prix de l'essence, pour renvoyer une valeur correspondant au coût d'un plein. Lorsque vous créez des fonctions VBA pour Excel, celles-ci sont accessibles pour l'utilisateur final comme n'importe quelle fonction Excel intégrée.

Les fonctions ont généralement besoin de *paramètres* ou *arguments*. Si les *arguments* obligatoires d'une fonction ne lui sont pas passés au moment de l'appel, une erreur est générée. Dans le cas précédent, trois paramètres de type numérique doivent être passés à la fonction `Coût_Plein` pour qu'elle s'exécute correctement : le contenu du réservoir, sa contenance et le prix de l'essence.

Le modèle d'objets d'Excel

Excel est l'application Office qui supporte VBA depuis le plus longtemps, et son modèle d'objets est le plus mûr. Excel offre de multiples possibilités de personnalisation au programmeur.

Les objets les plus importants sont présentés dans le tableau 1-2. Le listing 1-1

présente des exemples d'instructions VBA utilisant ces objets. L'essentiel de ces exemples a été généré à l'aide de l'Enregistreur de macro, sans qu'il soit nécessaire d'écrire du code.

Tableau 1-2. Les objets clés du modèle d'Excel

Collection (objet)	Description
Objets de niveau *Application*	
Add-ins (Add-in)	L'ensemble des macros complémentaires, chargées ou non. Accessibles dans la boîte de dialogue Macros complémentaires (Outils > Macros complémentaires)
Dialogs (Dialog)	Les boîtes de dialogue prédéfinies d'Excel
LanguageSettings	Renvoie des informations sur les paramètres de langue utilisés dans l'application.
Names (Name)	L'ensemble des objets Name de niveau Application. Un objet Name représente un nom défini pour une plage de cellules
Windows (Window)	L'ensemble des fenêtres disponibles (accessibles *via* le bouton Changer de fenêtre de l'onglet Affichage)
Workbooks (Workbook)	L'ensemble des classeurs ouverts
Worksheetfunction	On utilise l'objet Worksheetfunction pour accéder aux fonctions de feuilles de calcul à partir de VBA. Faites suivre Worksheetfunction d'un point, puis du nom de la fonction et de ses arguments entre parenthèses
Objets de l'objet *Workbook*	
Charts (Chart)	L'ensemble des feuilles graphiques de l'objet Workbook
Names (Names)	L'ensemble des objets Name pour le classeur spécifié
Styles (Style)	L'ensemble des styles disponibles dans un classeur. Il peut s'agir d'un style défini par l'utilisateur ou d'un style prédéfini, tel que Millier, Monétaire ou Pourcentage (Format > Styles)
Windows (Window)	L'ensemble des fenêtres pour le classeur spécifié
Worksheets (Worksheet)	L'ensemble des feuilles de calcul de l'objet Workbook désigné
Objets de l'objet *Worksheet*	
Names (Name)	L'ensemble des objets Name pour la feuille de calcul spécifiée
Range	Une cellule, une ligne, une colonne ou une plage de cellules, contiguës ou non, une plage de cellules 3D
Comments (Comment)	L'ensemble des commentaires pour l'objet Worksheet désigné
HPageBreaks (HPageBreak)	Les sauts de page horizontaux de la feuille de calcul
VPageBreaks (VPageBreaks)	Les sauts de page verticaux de la feuille de calcul
Hyperlinks (Hyperlink)	L'ensemble des liens hypertextes de la feuille de calcul
Scenarios (Scenario)	Les scénarios de la feuille de calcul
OLEObjects (OLEObject)	Les objets incorporés ou liés et les contrôles ActiveX de la feuille
Outline	Le plan de la feuille de calcul

PageSetup	Les options de mise en page de la feuille
QueryTables (QueryTable)	Les tables de requête de la feuille
PivotTables (PivotTable)	Les tableaux et les graphiques croisés dynamiques
ChartObjects (ChartObject)	Les graphiques incorporés de la feuille de calcul spécifiée
Objets de l'objet *Range*	
Areas	Les plages de cellules contiguës à l'intérieur d'une sélection
Borders (Border)	Les bordures d'un objet Range. La collection Borders regroupe toujours quatre objets Border, représentant les quatre bordures de l'objet Range désigné
Font	Les attributs de police de caractères de l'objet Range spécifié
Interior	L'intérieur de l'objet Range
Characters	L'ensemble des caractères contenus par l'objet Range
Name	Le premier nom dans la liste des noms de la plage de cellules précisée
Style	Le style de l'objet Range désigné
FormatConditions (FormatCondition)	L'ensemble des mises en forme conditionnelles de l'objet Range
Hyperlinks (Hyperlink)	L'ensemble des liens hypertextes de l'objet Range
Validation	La validation des données pour la plage de cellules précisée
Comment	Le commentaire de cellule pour l'objet Range désigné

Listing 1-1. Exemples d'utilisation des objets Excel

```
'activation du classeur Classeur1
Windows("Classeur1").Activate
'-----------------------------------------------------------
'sauvegarde du classeur actif
ActiveWorkbook.Save
'nouveau classeur
Workbooks.Add
'nouveau classeur fondé sur le modèle MonModele.xlt
Workbooks.Add("MonModele.xlt")
'-----------------------------------------------------------
'affectation du nom MaPlage à la plage de cellules A1:C20 de la feuille MaFeuille
ActiveWorkbook.Names.Add Name:= "MaPlage", RefersTo:= "=MaFeuille!$a$1:$c$20"
'ajout d'un graphique
Charts.Add
'affectation du type Histogramme empilé au graphique actif
ActiveChart.ChartType = xlColumnStacked
'définition de la source de données du graphique actif
ActiveChart.SetSourceData Source:=Sheets("Feuil1").Range("C6:E10"), PlotBy:=xlColumns
'définition de l'emplacement du graphique actif
ActiveChart.Location Where:=xlLocationAsObject, Name:= "Feuil1 "
'-----------------------------------------------------------
'sélection de la feuille Feuil1 du classeur actif
Sheets("Feuil1").Select
'affectation du nom Graphique à la feuille Feuil1
Sheets("Feuil1").Name = "Graphique"
'suppression des feuilles sélectionnées
ActiveWindow.SelectedSheets.Delete
'-----------------------------------------------------------
'ajout d'un commentaire à la cellule D2 de la feuille active
```

```
Range("D2").AddComment
’le commentaire n’est pas rendu visible
Range("D2").Comment.Visible = False
’définition du texte du commentaire de la cellule D2
Range("D2").Comment.Text Text:="Excellent !"
’-----------------------------------------------------------
’affectation du format monétaire US à la plage sélectionnée
Selection.NumberFormat = "#,##0.00 $"
’définition des attributs de police de la plage sélectionnée
With Selection.Font
 .Name = "Arial"
 .FontStyle = "Gras"
 .Size = 8
 .ColorIndex = 46
End With
’coloriage de l’intérieur de la plage sélectionnée
With Selection.Interior
 .ColorIndex = 6
 .Pattern = xlSolid
End With
```

2

Premières macros

L'enregistrement de macros constitue certainement le meilleur apprentissage de Visual Basic pour Applications. Les commandes de l'application hôte accessibles par les onglets ou les raccourcis clavier, le déplacement (à l'aide du clavier ou de la souris) dans un classeur et la modification de ce dernier peuvent être enregistrés dans une macro. Il suffit simplement de déclencher l'Enregistreur de macro et d'exécuter ces commandes, sans qu'il soit nécessaire d'écrire la moindre ligne de code. Ensuite, vous répétez autant de fois que vous le souhaitez la série d'instructions ainsi mémorisée en exécutant simplement la macro. Vous visualisez le *codage* de la macro dans la fenêtre Code de Visual Basic Editor. Vous découvrez ainsi la structure et la syntaxe des programmes VBA par la pratique.

Définition

Le code est le texte, écrit dans le langage de programmation, constituant le programme. Le codage désigne le fait de générer du code, soit en utilisant l'Enregistreur de macro, soit en l'écrivant directement dans la fenêtre de code de Visual Basic Editor.

À travers des exemples simples, ce chapitre vous initiera à l'enregistrement et à la création de macros. Vous créerez une première macro, puis en améliorerez très simplement la fonctionnalité. Vous verrez que cette méthode est relativement souple et qu'il existe plusieurs possibilités, plus ou moins efficaces et plus ou moins rapides, pour créer une macro. Vous apprendrez rapidement à utiliser l'une ou l'autre des méthodes disponibles (voire à les combiner), en fonction de l'objet de votre macro.

Créer une macro GrasItalique

Lorsque vous souhaitez enrichir le contenu d'une cellule d'attributs de caractères, une solution consiste à choisir le Format de cellule du bouton Format (onglet Accueil) et à sélectionner l'onglet Police. On définit ensuite les

attributs voulus et on valide en cliquant sur OK. Nous utiliserons ici cette méthode pour créer une macro enrichissant la cellule ou la plage de cellules active des attributs gras et italique.

Cette macro est fort simple, puisque composée de seulement deux commandes, mais elle aidera à découvrir comment les programmes VBA sont structurés. Le but de ce chapitre est de vous initier aux différentes méthodes de création et d'optimisation de macros. Prenez donc le temps de le lire dans sa totalité ; les principes acquis seront valables pour l'ensemble des macros que vous créerez par la suite, quel que soit leur niveau de complexité.

Afficher l'onglet Développeur

Avant toute chose, vous devez afficher l'onglet Développeur dans le ruban pour accéder aux fonctions de programmation VBA. Cliquez sur l'onglet Fichier du ruban, puis sur la commande Options. Dans la fenêtre Options Excel, sélectionnez Personnaliser le ruban. Cochez ensuite la case Développeur de la liste Onglets principaux (voir figure 2-1), puis validez. L'onglet Développeur apparaît sur le ruban.

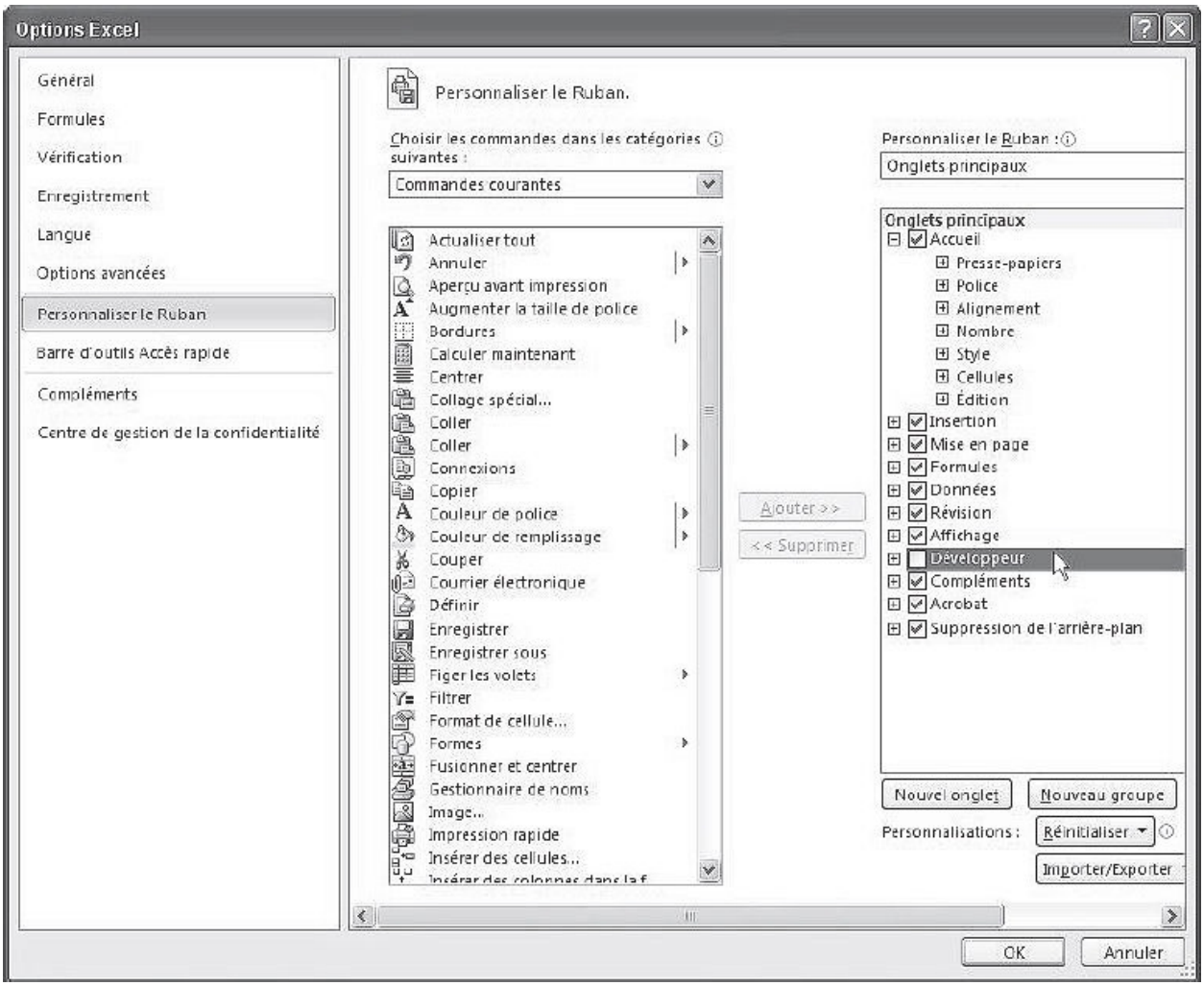


Figure 2-1 – *Activez l'onglet Développeur pour accéder aux fonctions de programmation du logiciel.*

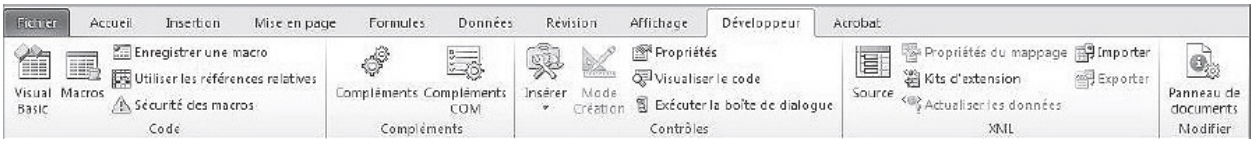


Figure 2-2 – *L'onglet Développeur est maintenant accessible sur le ruban.*

Démarrer l'enregistrement

Avant de commencer l'enregistrement de la macro GrasItalique, sélectionnez une cellule à laquelle vous attribuerez les formats de caractères voulus.

1. Cliquez sur le bouton Enregistrer une macro du groupe Code de l'onglet Développeur.

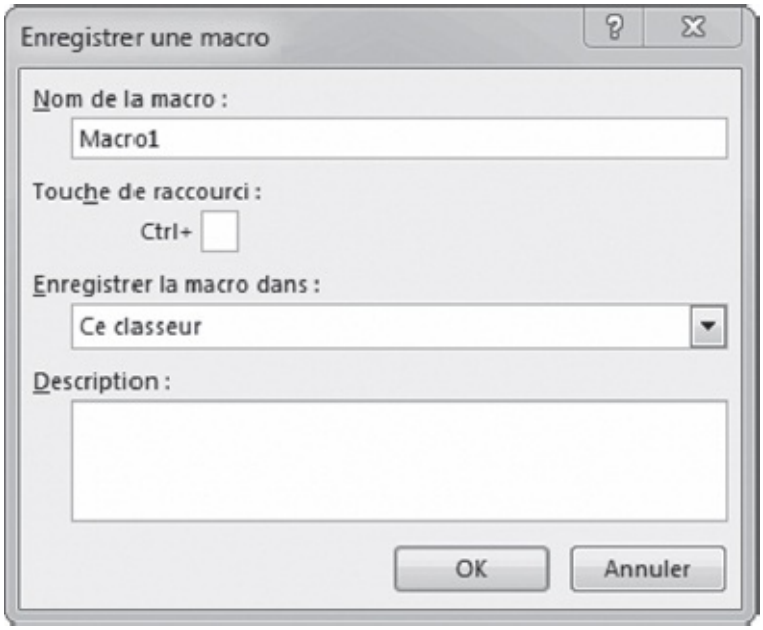


Figure 2-3 – *La boîte de dialogue Enregistrer une macro.*

2. Par défaut, la zone Nom de la macro indique Macro1. Remplacez ce nom par GrasItalique.

> **Conseil**
>
> Il est plus rapide d'enregistrer une macro sous le nom que lui attribue Excel par défaut. Cependant, si vous enregistrez plusieurs macros, celles-ci deviendront rapidement indiscernables. Attribuez-leur un nom représentatif et entrez une rapide description de la fonction de chacune dans la zone Description ; vous n'aurez ainsi aucun problème pour les distinguer.

3. Dans la zone Enregistrer la macro dans, choisissez Classeur de macros personnelles.
4. Dans la zone Description, tapez une brève description de la macro.
5. L'intérêt de la macro GrasItalique réside dans le gain de temps qu'elle apporte à l'utilisateur. L'attribution d'un raccourci clavier lui donnera donc toute son efficacité.

   Placez le curseur dans la zone de texte Touche de raccourci et saisissez une lettre qui, combinée à la touche Ctrl, sera affectée à l'exécution de la macro GrasItalique (dans notre exemple, la combinaison Ctrl+B). Vous pouvez aussi maintenir la touche Maj enfoncée de façon à affecter à votre macro une combinaison Ctrl+Maj+*Lettre*.

   La boîte de dialogue Enregistrer une macro doit maintenant se présenter comme à la figure 2-4.

Attention

Lorsque vous attribuez un raccourci clavier à une macro, aucune indication ne vous est fournie quant à l'affectation ou non de ce raccourci à une commande. Si le raccourci choisi était déjà affecté à une commande Excel, il sera réattribué à la macro sans que vous en soyez informé. Veillez donc à ne pas attribuer à votre macro un raccourci clavier déjà utilisé par Excel, particulièrement si d'autres utilisateurs sont amenés à utiliser vos macros.

Figure 2-4 – *La boîte de dialogue Enregistrer une macro complétée.*

6. Enfin, cliquez sur OK.

   Le libellé du bouton Enregistrer une macro devient Arrêter l'enregistrement, indiquant que la macro est en cours d'enregistrement.

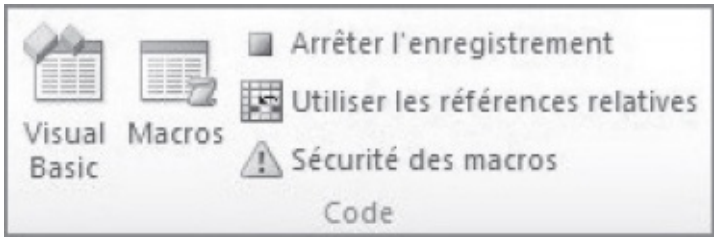


Figure 2-5 – *La commande Enregistrer une macro devient Arrêter l'enregistrement.*

Enregistrer les commandes de la macro

Comme nous l'avons dit au chapitre précédent, la création d'une macro simple ne nécessite pas la moindre ligne d'écriture. Il suffit d'exécuter les commandes qui la composent après avoir activé l'Enregistreur de macro : l'application hôte se charge de les convertir en langage Visual Basic.

Pour enregistrer la macro GrasItalique :

1. Activez l'onglet Accueil du ruban, puis cliquez sur le bouton Format situé dans l'angle inférieur droit du groupe Cellules, et sélectionnez la commande Format de cellule. Vous pouvez également utiliser le raccourci clavier Ctrl+Maj+F.
2. Dans la zone Style, sélectionnez Gras italique, puis cliquez sur OK.
3. Les commandes de la macro GrasItalique sont mémorisées. Cliquez sur le bouton Arrêter l'enregistrement de l'onglet Développeur.

Attention

Si vous sélectionnez une cellule après avoir déclenché l'Enregistreur de macro, cette manipulation sera enregistrée. Par conséquent, la macro appliquera la mise en forme Gras Italique à cette cellule et non aux cellules actives au moment de son exécution.

Exécuter la macro

Pour exécuter la macro GrasItalique, vous pouvez passer par la boîte de dialogue Macro ou – et c'est là que réside son intérêt – utiliser le raccourci clavier que nous lui avons attribué.

La boîte de dialogue Macro

1. Activez l'onglet Développeur du ruban, puis cliquez sur le bouton Macros du groupe Code. La boîte de dialogue Macro s'affiche.

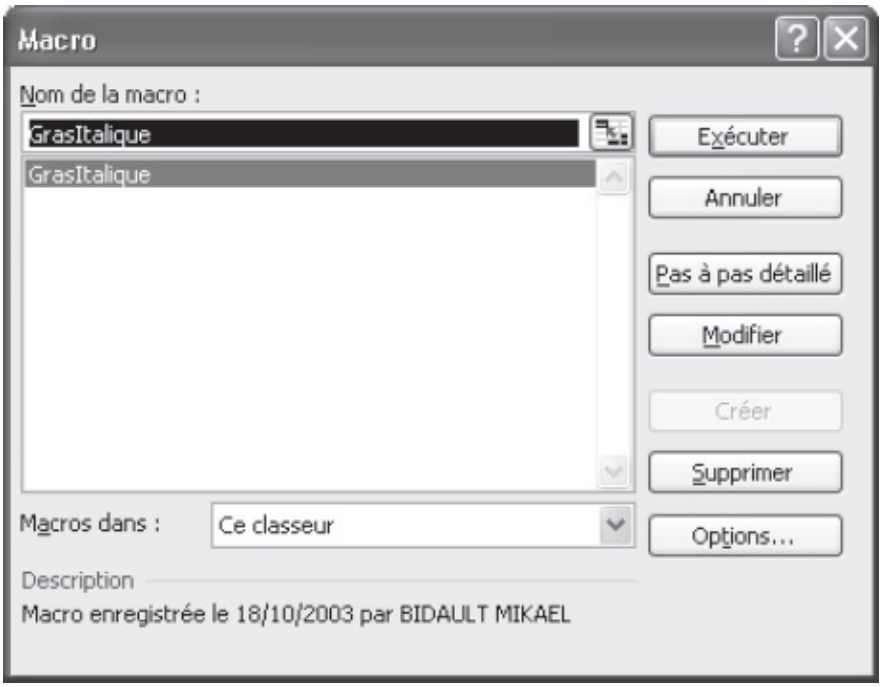


Figure 2-6 – *La boîte de dialogue Macro.*

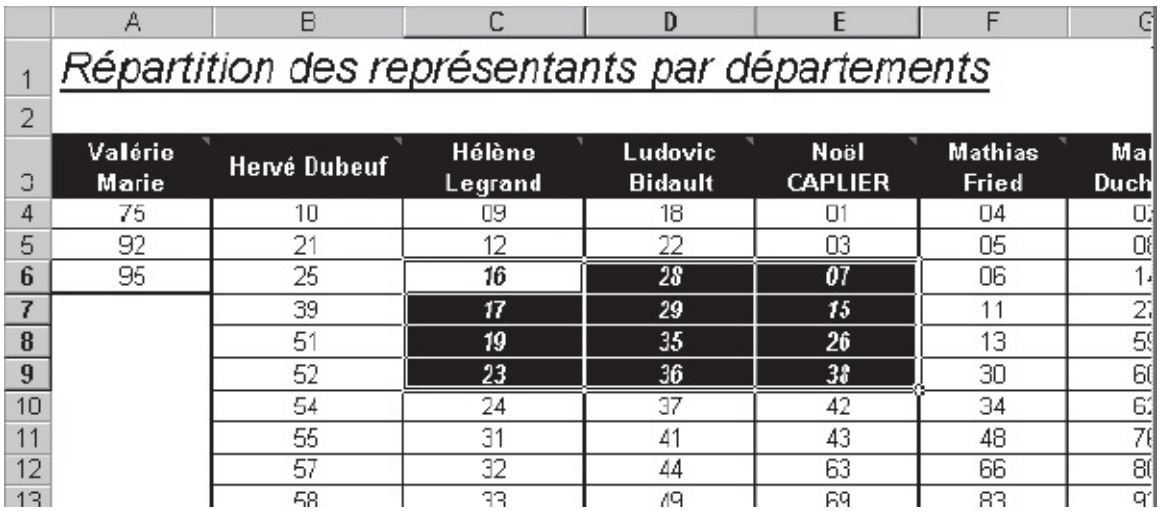
2. Dans la liste des macros disponibles, sélectionnez GrasItalique, qui s'affiche alors dans la zone Nom de la macro.
3. Cliquez sur le bouton Exécuter. La boîte de dialogue Macro disparaît automatiquement et les cellules sélectionnées s'enrichissent des attributs Gras et Italique (voir figure 2-7).

Si la procédure d'exécution que vous venez de mettre en œuvre convient à certaines macros plus complexes et d'un usage moins fréquent, elle ne présente pas d'intérêt pour GrasItalique puisqu'elle nécessite plus d'opérations pour l'utilisateur qu'elle n'en exécute.

Le raccourci clavier

1. Sélectionnez le texte voulu, puis tapez le raccourci clavier attaché à la macro (Ctrl+B).

En un clin d'œil, les cellules sélectionnées se sont enrichies des attributs de caractères voulus.



	A	B	C	D	E	F	G
1	*Répartition des représentants par départements*						
2							
3	Valérie Marie	Hervé Dubeuf	Hélène Legrand	Ludovic Bidault	Noël CAPLIER	Mathias Fried	Ma Duch
4	75	10	09	18	01	04	0
5	92	21	12	22	03	05	0
6	95	25	***16***	***28***	***07***	06	1
7		39	***17***	***29***	***15***	11	2
8		51	***19***	***35***	***26***	13	5
9		52	***23***	***36***	***38***	30	6
10		54	24	37	42	34	6
11		55	31	41	43	48	7
12		57	32	44	63	66	8
13		58	33	49	69	83	9

Figure 2-7 – *Les cellules sélectionnées après exécution de la macro.*

Structure de la macro

Lors de l'enregistrement de la macro, les actions que vous avez effectuées ont été converties en langage Visual Basic. Pour en visualiser la syntaxe :

1. Activez l'onglet Développeur, puis cliquez sur le bouton Macros. Dans la boîte de dialogue, sélectionnez la macro GrasItalique. Elle s'affiche dans la zone Nom de la macro.
2. Cliquez sur le bouton Modifier. Visual Basic Editor, l'environnement de développement intégré d'Office, s'ouvre sur la fenêtre Code de votre macro (voir figure 2-8).

Info

Lorsque vous tentez de modifier la macro, si Excel affiche le message « Impossible de modifier une macro dans un classeur masqué… », vous devez afficher le fichier PERSONAL.XLSB. Sélectionnez l'onglet Affichage, puis cliquez sur le bouton Afficher du groupe Fenêtre.

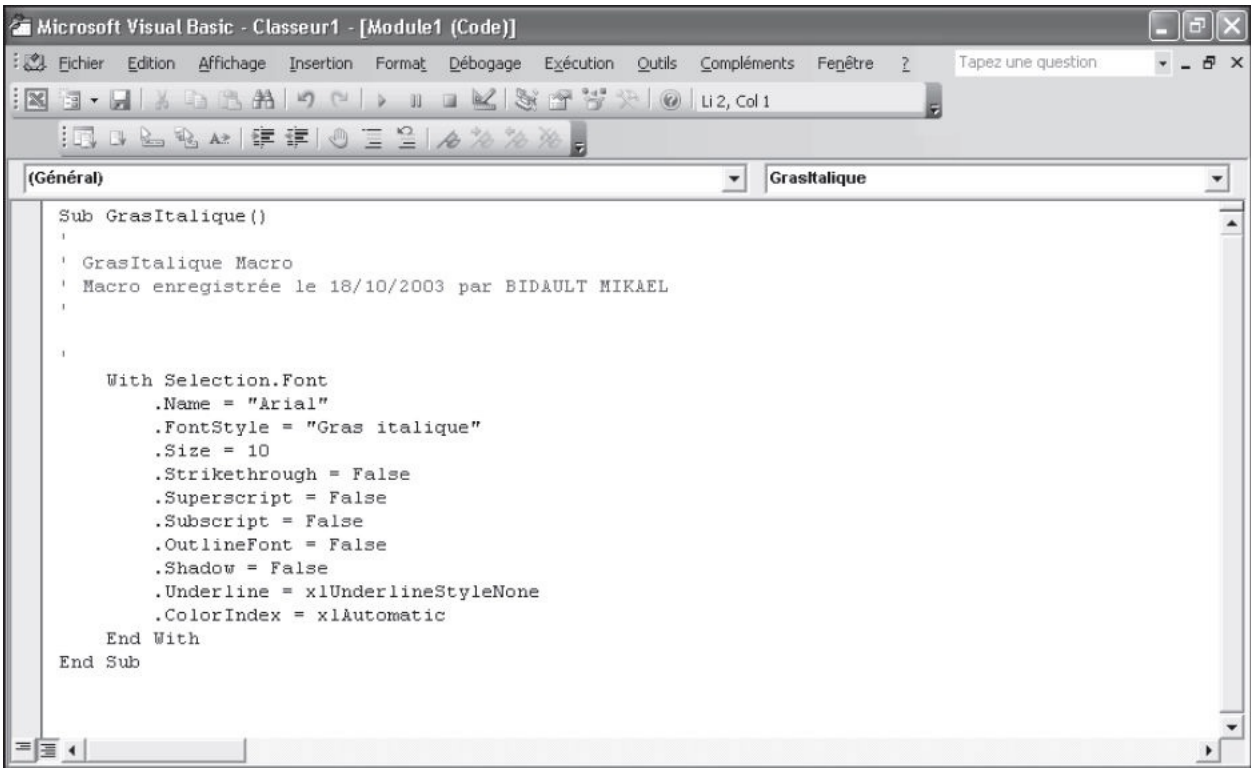


Figure 2-8 – *La fenêtre Code de Visual Basic Editor permet de visualiser et de modifier le code d'une macro.*

Examinons de plus près le texte de la macro GrasItalique. Il commence par l'instruction :

```
Sub GrasItalique()
```

et se termine par l'instruction :

```
End Sub
```

Ces deux instructions encadrent systématiquement toute macro enregistrée. `Sub` est l'abréviation de *subroutine* qui signifie *sous-routine* ou *sous-programme*. Les macros sont en effet des sous-programmes de l'application hôte. `GrasItalique()` est le nom de la sous-routine – c'est-à-dire le nom de la macro. `End Sub` indique la fin. Ces instructions sont indispensables au fonctionnement de toute macro. Cependant, vous n'aurez pas à vous en soucier : lorsque vous enregistrez une macro, les instructions qui la composent sont automatiquement encadrées par celles-ci.

Directement placées derrière l'instruction `Sub` *`NomMacro`*, des lignes de commentaires reprennent les informations que contenait la zone Description lors de l'enregistrement (voir figure 2-3), en l'occurrence le nom du créateur, la date d'enregistrement, ainsi que le texte descriptif de la macro.

Info

Les commentaires sont des indications ajoutées dans le code d'un programme afin d'en faciliter la lecture.

Certains éléments du code apparaissent en couleur. Cette mise en valeur distingue aisément les éléments constitutifs du code. Par défaut, Visual Basic Editor applique le vert aux commentaires et le bleu aux mots-clés du langage.

Définition

Un mot-clé est un mot ou un symbole reconnu comme élément du langage de programmation Visual Basic. Il peut s'agir d'une structure de contrôle, d'une fonction ou de tout autre élément du langage indépendant du modèle d'objets de l'application hôte. Les structures de contrôle sont des instructions qui servent à diriger le comportement d'une macro (par exemple, répéter une opération en boucle, n'effectuer une instruction que dans un contexte spécifique). Vous apprendrez à utiliser les structures de contrôle de Visual Basic au chapitres 7.

Entre les instructions `Sub GrasItalique()` et `End Sub` se trouvent les instructions qu'exécutera la macro :

```
With Selection.Font
        .Name = "Arial"
        .FontStyle = "Gras italique"
        .Size = 10
        .Strikethrough = False
        .Superscript = False
        .Subscript = False
        .OutlineFont = False
        .Shadow = False
        .Underline = xlUnderlineStyleNone
        .ThemeColor = xlThemeColorLight1
        .TintAndShade = 0
        .ThemeFont = xlThemeFontMinor
End With
```

Info

Si vous utilisez une version antérieure à Excel 2007, les trois lignes `ThemeColor`, `TintAndShade` et `ThemeFont` sont remplacées par une seule ligne de code :

```
.ColorIndex = xlAutomatic
```

Il s'agit des commandes effectuées lors de l'enregistrement : ces lignes indiquent à la macro les actions à accomplir. Leur structure peut vous dérouter, mais vous vous y habituerez rapidement :

- L'expression `Selection.Font` indique à la macro qu'il s'agit d'appliquer un format de police aux cellules sélectionnées :
  - `Selection` est une propriété qui renvoie un objet `Selection` représentant la

sélection en cours dans le document actif. Lorsque vous enregistrerez des macros, vous verrez que certains objets, et les propriétés qui leur sont associées, sont très usités. C'est le cas de `Selection`, qui apparaît dans le code d'une macro Excel chaque fois qu'une opération (format de police, dimensions, définition d'une catégorie de données, etc.) est effectuée sur une plage de cellules sans que celle-ci soit définie auparavant ;

  - `Font` indique à la macro qu'il s'agit d'appliquer un format de police à l'objet `Selection` (les cellules sélectionnées).

- Les instructions `With` et `End With` encadrent l'ensemble des propriétés de l'objet `Font`. Comme tout mot-clé, elles apparaissent en bleu dans la fenêtre de code. Lorsque, durant l'enregistrement d'une macro, vous faites appel à une boîte de dialogue dans laquelle plusieurs options sont définies, cette structure est utilisée pour *coder* l'ensemble des options de la boîte de dialogue, selon la syntaxe suivante :

```
With Objet
  Propriétés de l'objet
End With
```

Définition

Le verbe « coder » désigne la transcription d'actions propres à l'application dans un langage de programmation déterminé.

- Chaque ligne située entre les instructions `With Selection.Font` et `End With` correspond à une option de l'onglet Police lors de l'enregistrement de la macro. Il s'agit des *propriétés* de l'objet `Font`. Remarquez que les propriétés sont toujours précédées d'un point.

  À chaque propriété est affectée une *valeur*. Elle indique l'état de cette option lors de l'enregistrement de la commande. Cette valeur peut être :

  - **`False`** ou **`True`** (valeur booléenne). Indiquent respectivement que l'option n'était pas cochée (faux) ou qu'elle l'était (vrai). `Superscript = False` indique ici que la case à cocher Exposant était décochée.

Rappel

Vous pouvez aussi utiliser les valeurs `-1` et `0` à la place de `True` et `False`. Par exemple, l'expression `.Superscript = True` pourra être remplacée par `.Superscript = -1`.

  - Une chaîne de caractères. Lorsqu'une propriété est attachée à une chaîne

de caractères, cette valeur est placée entre guillemets. `Name = "Arial"` indique le nom de la police en cours dans la boîte de dialogue Police lors de l'enregistrement de la macro.

– Une valeur numérique. Les valeurs possibles varient d'une propriété à l'autre. `Size = 10` indique ici que le corps de la police est de 10 points. Dans Excel, cette valeur doit être définie entre 1 et 409.

– Une constante. Il s'agit d'une valeur prédéfinie qui permet de paramétrer une propriété. Par exemple, la propriété `Underline` définit le type de soulignement appliqué à la police ou à la plage. Sa valeur correspond à l'état de l'option Soulignement lors de l'enregistrement de la macro. Elle est ici attachée à la constante `xlUnderlineStyleNone`, qui correspond à l'option Aucun de la liste déroulante Soulignement. Il existe une constante `XlUnderlineStyle` spécifique pour chaque option de cette liste (`xlUnderlineStyleDouble` pour Soulignement double, `xlUnderlineStyleSingle` pour Soulignement simple, etc.).

Le tableau 2-1, en présentant à quelles options de la boîte de dialogue Format de cellule (onglet Police) les propriétés de l'objet `Font` sont associées, vous aidera à comprendre comment les actions que vous enregistrez dans la macro sont codées par Visual Basic pour Applications.

Tableau 2-1. Les propriétés de l'objet Font d'Excel

Propriété	Format de cellule (onglet Police)	Valeurs autorisées
`Name`	Zone de texte Police	Chaîne de caractères correspondant au nom d'une police disponible dans la zone de liste modifiable [1].
`FontStyle`	Zone de texte Style	Chaîne de caractères correspondant à l'option sélectionnée dans la zone de liste [1].
`Size`	Zone de texte Taille	Valeur numérique représentant le corps de la police. Cette valeur peut être comprise entre 1 et 409 [1].
`Strikethrough`	Case à cocher Barré	`True` (barré) ou `False` (non barré) [1].
`Superscript` `SuperScript`	Case à cocher Exposant	`True` (mise en forme exposant) ou `False` (pas de mise en forme exposant) [1]. Les attributs Exposant et Indice ne pouvant être appliqués à une même sélection, lorsque vous affectez la valeur `True` à la propriété `SuperScript`, la propriété `Subscript` prend la valeur `False`[2].
		`True` (mise en forme indice) ou `False` (pas de mise en forme

`Subscript` `SubScript`	Case à cocher Indice	indice) [2]. Les attributs Indice et Exposant ne pouvant être appliqués à une même sélection, lorsque vous affectez la valeur `True` à la propriété `SubScript`, la propriété `SuperScript` prend la valeur `False`[2].
`OutlineFont` et `Shadow`	[Aucune correspondance]	`True` ou `False` (sans effet). Ces propriétés indiquent respectivement si la police possède une mise en forme Relief et Ombré. Elles ne correspondent à aucune option de la boîte de dialogue Format de cellule, mais ont été conservées comme propriétés de l'objet `Font` d'Excel. Elles sont sans effet sur la police.
`Underline`	Liste déroulante Soulignement	Une des cinq constantes `xlUnderlineStyleNone` représentant les cinq types de soulignement disponibles dans Excel [1].
`ThemeColor`, `TintAndShade` et `ThemeFont`	Onglet Remplissage	Respectivement une des constantes `xlThemeColor` (la couleur de motif), une valeur numérique comprise entre –1 et 1 représentant la teinte appliquée à cette couleur (de sombre à lumineux) et une des constantes `xlThemeFont` qui correspond à la police du thème.

1. Si vous demandez la valeur d'une propriété pour une plage contenant des cellules dont les attributs correspondants sont différents, la valeur Null sera renvoyée. Par exemple, si vous cherchez la valeur de la propriété Name de l'objet Font d'une plage de cellules contenant à la fois des cellules en police Arial et d'autres en police Times, la valeur Null sera renvoyée.
2. Notez que cet état est le reflet de ce qui se passe dans la boîte de dialogue Police. En effet, vous ne pouvez pas cocher à la fois l'option Indice et l'option Exposant.

Comme le montre le tableau précédent, les actions exécutées sont codées selon des principes récurrents auxquels l'enregistrement de macros vous familiarisera.

La macro GrasItalique ouvre donc (virtuellement) la boîte de dialogue Format de cellule sur l'onglet Police et y définit les options telles qu'elles l'ont été lors de l'enregistrement. Elle applique ensuite ces propriétés au texte sélectionné.

Fermez la fenêtre Visual Basic Editor, en sélectionnant la commande Fermer et retourner à Microsoft Excel du menu Fichier.

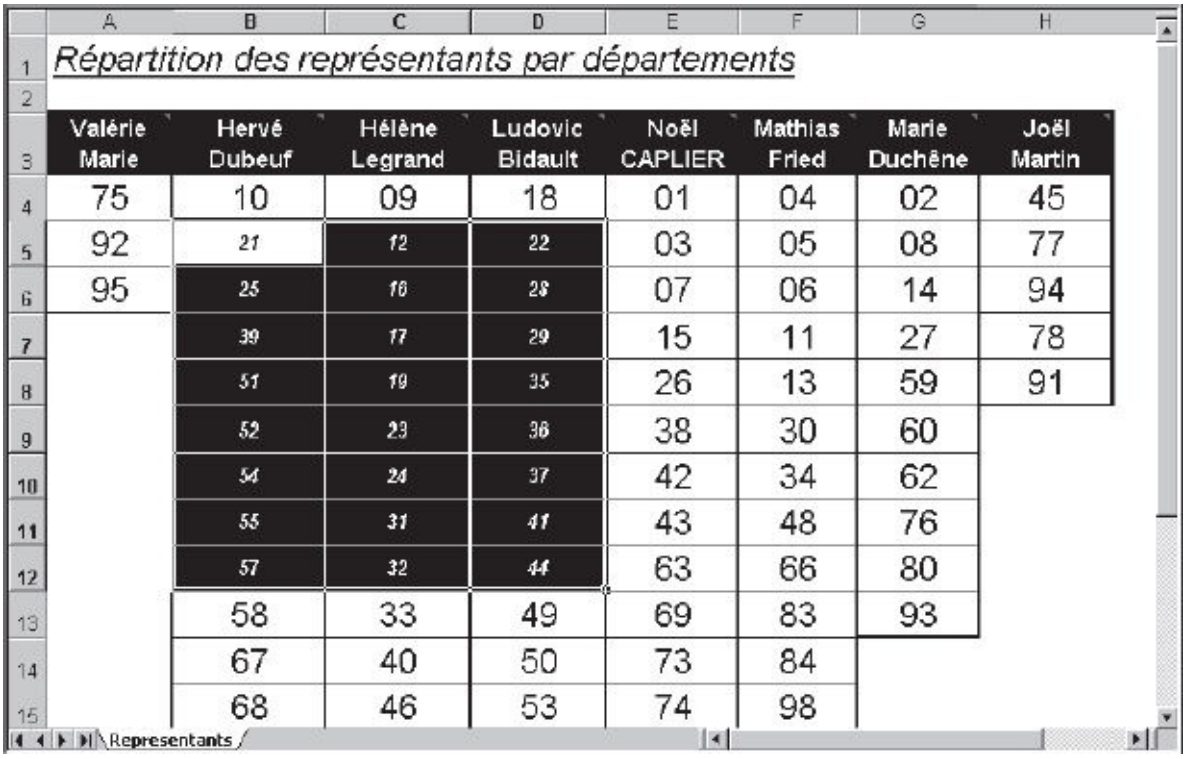
Améliorer la macro

Sélectionnez maintenant une cellule dont la police et le corps sont différents de ceux de la plage sélectionnée lors de l'enregistrement de la macro. Tapez le raccourci clavier affecté à la macro (Ctrl+B). Celle-ci s'exécute.

À la figure 2-9, on constate que les attributs Gras et Italique ont bien été appliqués, mais la police et le corps du texte ont changé. Tous les arguments en

cours dans la boîte de dialogue Police ont en effet été pris en compte lors de l'enregistrement de la macro.

Cela apparaît clairement dans la fenêtre de code (voir figure 2-7) : les arguments `.Size = 10` et `.Name = "Arial"` correspondent à la police et au corps du texte sélectionnés lors de l'enregistrement de la macro.



Répartition des représentants par départements

Valérie Marie	Hervé Dubeuf	Hélène Legrand	Ludovic Bidault	Noël CAPLIER	Mathias Fried	Marie Duchêne	Joël Martin
75	10	09	18	01	04	02	45
92	21	12	22	03	05	08	77
95	25	16	28	07	06	14	94
	39	17	29	15	11	27	78
	51	19	35	26	13	59	91
	52	23	36	38	30	60	
	54	24	37	42	34	62	
	55	31	41	43	48	76	
	57	32	44	63	66	80	
	58	33	49	69	83	93	
	67	40	50	73	84		
	68	46	53	74	98		

Representants

Figure 2-9 – *L'ensemble des arguments en cours lors de l'enregistrement de la macro est appliqué.*

Pour remédier à ce problème, supprimez les attributs indésirables directement à partir de la fenêtre de code de la macro :

1. Activez l'onglet Développeur du ruban, puis cliquez sur le bouton Macros du groupe Code. Sélectionnez GrasItalique, puis cliquez sur le bouton Modifier.
2. Supprimez toutes les propriétés de l'objet `Font` que la macro ne doit pas modifier (toutes les instructions sauf `.FontStyle = "GrasItalique"`). Dans le menu Fichier, choisissez Enregistrer PERSONAL.XLSB ou cliquez sur le bouton Enregistrer de la barre d'outils Standard.

   Le texte de la macro doit se présenter ainsi :

```
Sub GrasItalique()
  With Selection.Font
```

```
      .FontStyle = "Gras italique"
   End With
End Sub
```

Choisissez Fichier > Fermer. Le tour est joué.

Info

La structure `With…End With` est utilisée pour paramétrer les propriétés d'un objet sans avoir à répéter la référence à cet objet pour chaque propriété. Puisque la macro ne définit ici qu'une propriété, il est inutile d'utiliser cette structure. La macro se présente alors ainsi :

```
Sub GrasItalique()
   Selection.Font.FontStyle = "Gras italique"
End Sub
```

Au fur et à mesure que vous avancerez dans l'apprentissage de la programmation Excel, vous découvrirez par la pratique les différents éléments des boîtes de dialogue Macro et Enregistrer une macro. Le tableau 2-2 en présente rapidement les fonctions.

Tableau 2-2. Fonctions des boîtes de dialogue Macro et Enregistrer une macro

Bouton	Description
Boîte de dialogue Macro	
Exécuter	Exécute la macro sélectionnée – dont le nom apparaît dans la zone de texte Nom de la macro.
Annuler	Ferme la boîte de dialogue Macro.
Pas à pas détaillé	Ouvre la fenêtre de code de la macro sélectionnée dans Visual Basic Editor et l'exécute étape par étape (instruction par instruction). Cette commande constitue un précieux outil de débogage.
Modifier	Ouvre la fenêtre de code de la macro sélectionnée dans Visual Basic Editor afin d'en permettre la modification.
Créer	Ouvre, dans Visual Basic Editor, une fenêtre Code simplement composée des instructions `Sub NomMacro()` et `End Sub`. Pour accéder à ce bouton, il faut auparavant saisir un nom de macro dans la zone Nom de la macro. Ce nom ne peut être le même que celui d'une macro existante.
Supprimer	Supprime la macro sélectionnée. Un message de confirmation s'affiche.
Options	Ouvre la boîte de dialogue Options de macro pour la macro sélectionnée, permettant de lui attribuer un raccourci clavier et d'en modifier la description.
Nom de la macro	Permet de désigner une macro existante ou de saisir le nom d'une nouvelle macro. Lorsque vous en sélectionnez une, son nom s'affiche dans cette zone de texte.
Macros dans	Désigne le classeur dont vous souhaitez afficher les macros [1].
Boîte de dialogue Enregistrer une macro	
	Nom de la macro qui sera enregistrée. Si le nom spécifié est déjà attribué à une

Nom de la macro	macro existante, l'application hôte affichera une boîte de dialogue vous demandant de confirmer le remplacement de l'ancienne.
Touche de raccourci	Affecte un raccourci clavier à la macro que l'on souhaite enregistrer.
Enregistrer la macro dans	Désigne le classeur où sera stockée la macro [1]. Le lieu de stockage détermine à partir de quels documents la macro sera disponible, c'est-à-dire où elle pourra être exécutée, modifiée ou supprimée.
Description	Destinée à la saisie d'une description de la macro. Par défaut, la date de création et le créateur apparaissent dans cette zone.
Bouton OK	Démarre l'enregistrement de la macro sans qu'aucun raccourci ne lui soit attribué.
Bouton Annuler	Ferme la boîte de dialogue sans déclencher l'Enregistreur de macro.

1. Le stockage et la disponibilité des macros sont traités à la fin de ce chapitre.

Une autre méthode d'enregistrement

L'Enregistreur de macro est un instrument souple qui mémorise l'ensemble des commandes que vous exécutez, en utilisant n'importe laquelle des méthodes que propose l'application hôte pour ce faire.

Dans le cas de la macro GrasItalique, il est plus simple de cliquer successivement sur les icônes Gras et Italique de la barre d'outils que de passer par la boîte de dialogue Format de cellule. Rien ne vous empêche d'enregistrer votre macro de la même façon.

Enregistrement

Pour réenregistrer la macro GrasItalique :

1. Sélectionnez une cellule.
2. Cliquez sur le bouton Enregistrer une macro de l'onglet Développeur. Dans la zone Nom de la macro de la boîte de dialogue, saisissez GrasItalique.
3. Affectez un raccourci clavier à la macro et saisissez une brève description.
4. Cliquez sur le bouton OK. Une boîte de dialogue s'affiche, vous demandant de confirmer le remplacement de la macro existante. Confirmez.

Figure 2-10 – *Confirmez le remplacement de la macro GrasItalique.*

5. Cliquez tour à tour sur les icônes Gras et Italique de la barre d'outils.
6. Cliquez sur le bouton Arrêter l'enregistrement.

   La macro est enregistrée.

Vous pouvez aussi enregistrer la macro en utilisant les raccourcis clavier affectés aux enrichissements Gras et Italique – respectivement Ctrl+G et Ctrl+I.

Structure de la macro

Observons la façon dont ces actions ont été codées en Visual Basic. Ouvrez la fenêtre de code de GrasItalique (bouton Macro de l'onglet Développeur, puis Modifier).

Le texte de la macro se présente ainsi :

```
Sub GrasItalique()
    Selection.Font.Bold = True
    Selection.Font.Italic = True
End Sub
```

Les propriétés `Bold` et `Italic` de l'objet `Font` sont définies à `True`, indiquant que les cellules sélectionnées seront enrichies des attributs gras et italique.

Remarquez l'absence de la structure `With…End With`. Cette structure n'est utilisée que lorsque plusieurs propriétés d'un objet sont validées dans une seule action – c'est le cas pour toutes les options d'une boîte de dialogue au moment où vous cliquez sur le bouton OK.

Vous pouvez cependant utiliser cette structure afin d'améliorer la lisibilité de la macro. Elle doit alors se présenter ainsi :

```
Sub GrasItalique()
    With Selection.Font
        .Bold = True
        .Italic = True
    End With
End Sub
```

Écrire la macro

Maintenant que vous connaissez la structure de la fenêtre de code, vous allez écrire directement la macro, sans l'aide de votre programmeur attitré, l'Enregistreur de macro.

Pour écrire GrasItalique :

1. Cliquez sur le bouton Macros de l'onglet Développeur. Sélectionnez GrasItalique. Cliquez sur le bouton Supprimer. Confirmez la suppression.
2. Cliquez de nouveau sur le bouton Macros de l'onglet Développeur. Dans la zone Nom de la macro, saisissez GrasItalique, puis cliquez sur le bouton Créer. Visual Basic Editor s'ouvre sur la fenêtre de code.

   Le texte de la macro se présente sous sa forme minimale :

```
Sub GrasItalique()
End Sub
```

3. Insérez une ligne entre `Sub GrasItalique()` et `End Sub`. Saisissez simplement le texte de la macro tel que nous l'avons vu lors de la section précédente.
4. Dans le menu Fichier, choisissez Enregistrer PERSONAL.XLSB, puis Fermer et retourner dans Microsoft Excel.

Créer une macro n'est pas plus compliqué que cela.

Contrairement à la méthode de l'enregistrement, la *création* d'une macro ne permet pas l'attribution d'un raccourci clavier. Pour y remédier, procédez comme suit :

1. Choisissez Outils > Macro > Macros ou, si vous utilisez Excel 2007, cliquez sur le bouton Macros de l'onglet Développeur.
2. Sélectionnez GrasItalique, puis cliquez sur le bouton Options.
3. Dans la boîte de dialogue qui s'affiche, indiquez un raccourci clavier et saisissez éventuellement une description pour la macro (voir figure 2-11). Validez en cliquant sur OK.

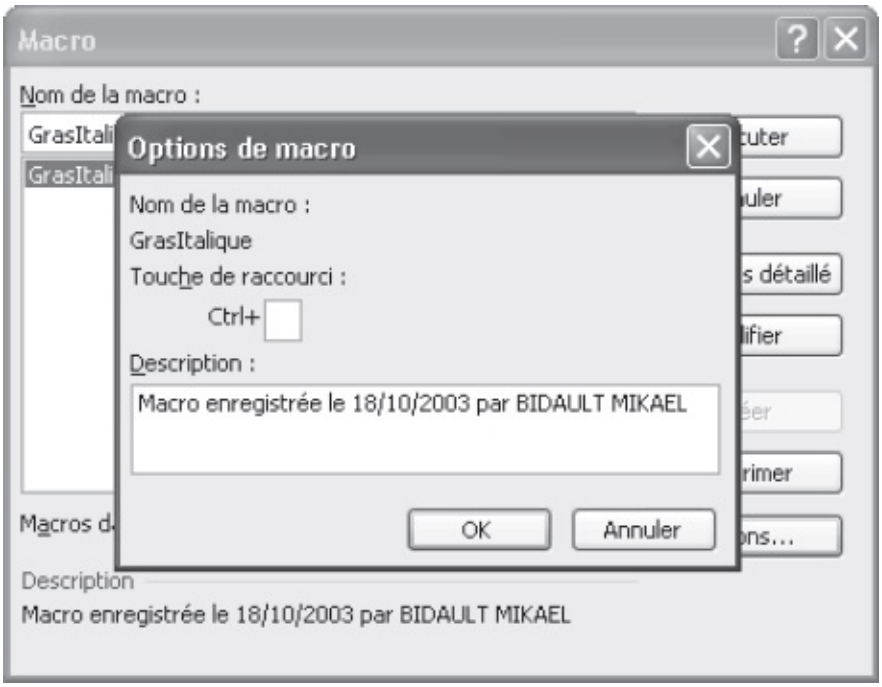


Figure 2-11 – *La boîte de dialogue Options de macro permet d'affecter un raccourci clavier à une macro existante.*

Info

Visual Basic pour Applications n'impose pas la saisie des majuscules. Celles-ci sont placées dans le code dans le seul but d'en faciliter la lecture. Vous pouvez parfaitement saisir du texte entièrement en minuscules (`selection.font`) dans une fenêtre de code.

Astuce

Si vous saisissez du texte en minuscules dans une fenêtre de code, Visual Basic replace les majuscules dans les instructions qu'il reconnaît lorsque vous changez de ligne. S'il ne modifie pas la casse d'une instruction saisie en minuscules, c'est qu'il ne la reconnaît pas. Par exemple, `selection.font.bold = true` deviendra `Selection.Font.Bold = True` lors du changement de ligne ; en revanche, si vous tapez `selection.font.old = true`, Word ne placera pas de capitale à `old`. C'est un bon moyen de vérifier que vous n'avez pas commis de fautes lors de la saisie.

Exécution de la macro

Une macro *créée* s'exécute exactement de la même façon qu'une macro *enregistrée*. Vous pouvez exécuter la macro GrasItalique, soit à partir de la boîte de dialogue Macros, soit en utilisant le raccourci clavier que vous lui aurez attribué après coup.

Vous savez maintenant enregistrer (selon la méthode de votre choix) et créer

une macro. Si GrasItalique vous paraît anodine, sachez que les principes acquis ici sont valables pour toutes les macros, quelle que soit l'application hôte.

GrasItalique est une véritable commande que vous avez ajoutée à Excel. En procédant de la même façon, vous pouvez créer n'importe quelle commande, en fonction de vos besoins.

Info

Vous avez appris dans ce chapitre à enregistrer et à créer une macro. La mise en œuvre de macros complexes nécessite souvent de combiner ces deux méthodes. On enregistre en général les commandes de la macro, puis on y écrit les fonctions qui ne peuvent être enregistrées.

Astuce

Pour qu'une macro s'exécute automatiquement à l'ouverture d'un classeur, affectez-lui le nom Auto_Open. Cette fonction est intéressante si vous souhaitez paramétrer différemment Excel selon les classeurs affichés. Enregistrez simplement les options d'Excel dans une macro Auto_Open.

Pour qu'une macro s'exécute automatiquement à la fermeture d'un classeur, affectez-lui le nom Auto_Close. Vous pouvez ainsi mettre à jour un autre fichier, créer une sauvegarde du fichier dans un autre dossier, afficher un message à l'attention de l'utilisateur, etc.

Choisir l'accessibilité des macros

Lorsque vous enregistrez ou créez des macros, celles-ci sont stockées dans un *projet* VBA, attaché à un document spécifique de l'application hôte. Pour exécuter une macro, le document hébergeant le projet doit être actif. Le stockage des macros est donc une donnée fondamentale, puisqu'il en détermine l'accessibilité pour l'utilisateur final. Une bonne gestion des macros est le préalable à une application puissante et efficace.

Accessibilité globale ou limitée

Une macro peut être accessible – c'est-à-dire exécutée, modifiée, renommée ou supprimée – à partir de n'importe quel document, ou limitée à des documents spécifiques. S'il est intéressant de pouvoir assurer une accessibilité globale aux macros, il est parfois préférable d'attacher une macro à un classeur spécifique. C'est le cas si la macro est conçue pour fonctionner avec un certain type de données et si elle est inutile dans d'autres classeurs – voire susceptible d'y provoquer des dommages. En outre, limiter la disponibilité des macros aux documents concernés en facilite la gestion.

Le stockage d'une macro est défini lors de son enregistrement ou de sa

création ; il est modifiable par la suite. Une macro peut également être créée directement à partir de Visual Basic Editor, sans passer par les boîtes de dialogue Macros ou Enregistrer une macro. Son stockage est alors déterminé dans l'Explorateur de projet. Ce dernier et la création à partir de Visual Basic Editor sont respectivement présentés aux chapitres 4 et 5.

Lors de l'enregistrement d'une macro, son affectation à un document s'effectue par la zone de liste Enregistrer la macro dans (voir figure 2-12). Les sections suivantes présentent les possibilités de stockage des macros Excel.

Figure 2-12 – *Sélectionnez le document de stockage de la macro lors de son enregistrement.*

Classeurs et modèles

Les macros enregistrées dans Excel sont stockées dans des classeurs ou dans des modèles. Pour accéder à une macro, il faut que le classeur dans lequel elle est stockée soit ouvert. Si plusieurs classeurs sont ouverts, vous pouvez accéder aux macros de l'un d'entre eux à partir de n'importe quel autre classeur.

Les macros enregistrées dans un modèle sont accessibles lorsque vous créez un nouveau classeur fondé sur ce modèle (en choisissant la commande Nouveau de l'onglet Fichier et en sélectionnant un modèle). Lorsque vous enregistrez le nouveau classeur, les macros du modèle sont « copiées » dans celui-ci et restent donc disponibles par la suite, lorsque vous rouvrez le classeur.

Notez cependant que les classeurs Excel n'entretiennent pas de lien avec le modèle à partir duquel ils ont été créés. Si vous ajoutez, modifiez ou

supprimez des macros dans un modèle, ces changements ne seront pas effectifs pour les classeurs préalablement créés à partir du modèle.

Pour enregistrer ou créer une macro dans un modèle, vous devez ouvrir le modèle en question.

Les classeurs dans lesquels sont stockées les macros sont identifiés par une extension spécifique : .xltm ou .xlsm.

Le classeur de macros personnel

Lors de l'enregistrement, vous pouvez choisir de stocker la macro dans le classeur actif afin d'en limiter la disponibilité à ce dernier. Elle ne pourra alors être exécutée, modifiée ou supprimée qu'à condition que ce classeur soit ouvert. Toutefois, vous aurez certainement besoin d'accéder à la plupart de vos macros à partir de classeurs différents. Pour qu'elles soient accessibles à partir de n'importe quel classeur Excel, il suffit de les enregistrer dans le classeur de macros personnel, PERSONAL.XLSB.

PERSONAL.XLSB est créé la première fois que vous enregistrez une macro dans le classeur de macros personnel, et la boîte de dialogue présentée à la figure 2-13 s'affiche alors.

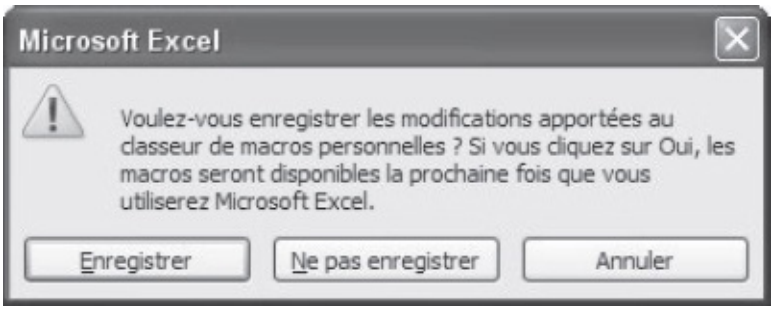

Microsoft Excel
Voulez-vous enregistrer les modifications apportées au classeur de macros personnelles ? Si vous cliquez sur Oui, les macros seront disponibles la prochaine fois que vous utiliserez Microsoft Excel.
Enregistrer
Ne pas enregistrer
Annuler

Figure 2-13 – *Le classeur de macros personnel est créé lorsque vous quittez Excel après y avoir enregistré ou écrit votre première macro.*

Le classeur de macros personnel est ouvert chaque fois que vous exécutez Excel. Vous pouvez donc en exécuter les macros qui y sont stockées à partir de n'importe quel classeur.

Info

Par défaut, le classeur de macros personnel est masqué au lancement d'Excel. Pour y accéder, choisissez la commande Afficher de l'onglet Affichage et sélectionnez PERSONAL.XLSB dans la boîte de dialogue Afficher.

Conseil

Le classeur de macros personnel est stocké dans le dossier XLSTART. Ce classeur contiendra probablement l'essentiel de vos macros. Il est donc conseillé d'en effectuer régulièrement une sauvegarde. Par défaut, l'emplacement de ce fichier est :

\Users*Nom_utilisateur*\AppData\Roaming\Microsoft\Excel\XLSTART

Les macros complémentaires

Les macros peuvent également être attachées à un classeur enregistré en tant que complément Excel (extension XLAM). Ce type de classeur est particulièrement adapté à la distribution de macros. Les compléments peuvent en effet être « chargés » dans Excel. Les macros qui y sont contenues sont alors rendues accessibles au lancement de l'application. Contrairement au classeur de macros personnel, les compléments ne sont pas « ouverts » : les macros qu'ils contiennent sont chargées en mémoire et les onglets d'Excel sont mis à jour pour intégrer les fonctionnalités qu'elles apportent.

L'enregistrement au format de complément est indéniablement la solution adaptée si vous développez des solutions complètes par modules. Vous évitez ainsi de surcharger le classeur de macros personnel et pouvez regrouper les macros par classeur, tout en leur assurant une accessibilité globale. Les principaux avantages des macros complémentaires sont les suivants :

- distribution et gestion simplifiées ;
- possibilité d'activation/désactivation très simple ;
- économie de ressources mémoire ;
- exécution plus rapide des macros ;
- ajout de commandes à l'application de façon transparente pour l'utilisateur ;
- les macros complémentaires chargées n'apparaissent pas dans la liste des macros.

Enregistrer un complément Excel

Pour enregistrer un classeur en tant que complément, commencez par préparer le projet VBA lui-même. Accédez à Visual Basic Editor :

1. Commencez par vous assurer que vos programmes VBA fonctionnent correctement et ne contiennent pas de bogues.
2. Dans Visual Basic Editor, ouvrez un module du projet et choisissez

Débogage > Compiler.

3. Protégez éventuellement votre projet par mot de passe (voir chapitres 16).
4. Quittez ensuite Visual Basic Editor et, dans Excel, affichez les Propriétés du fichier en cliquant sur l'onglet Fichier, puis sur Informations. Dans la partie droite de la fenêtre qui s'affiche, cliquez sur le bouton Propriétés, puis choisissez Propriétés avancées.
5. Activez l'onglet Résumé de la boîte de dialogue Propriétés. Saisissez un nom et un descriptif représentatifs dans les zones Titre et Commentaire (voir figure 2-14). Ils apparaîtront dans la boîte de dialogue Macros complémentaires.

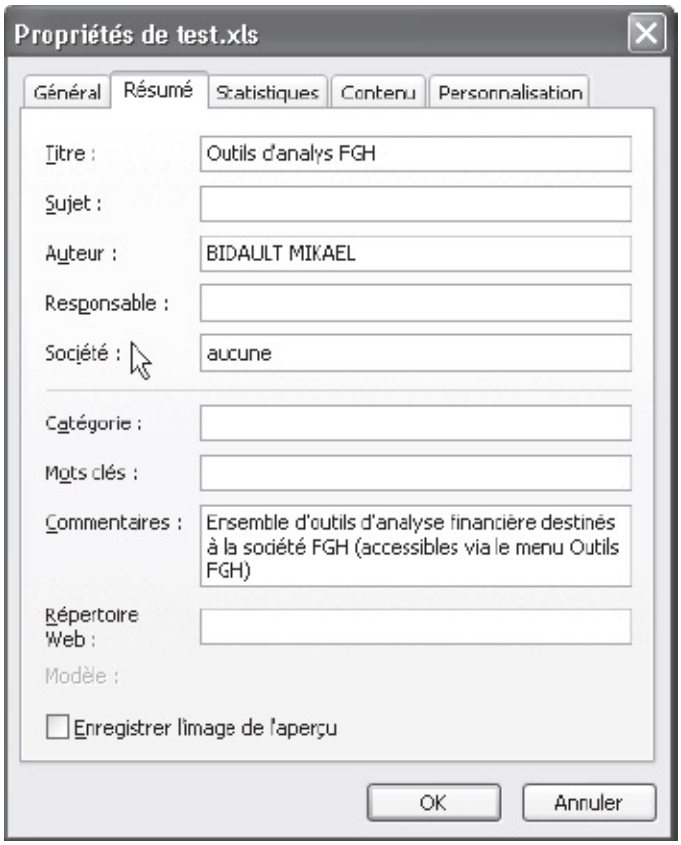


Figure 2-14 – *Choisissez un titre et un commentaire clairs.*

6. Choisissez la commande Enregistrer sous. Dans la zone Type de fichier, sélectionnez Complément Excel (*.xlam). Le dossier *AddIns* est activé par défaut.
7. Affectez un nom adapté et cliquez sur Enregistrer.

   Le complément est enregistré et automatiquement fermé.

Activer/désactiver un complément

Pour activer ou désactiver une macro complémentaire, procédez comme suit :

1. Cliquez sur l'onglet Fichier, puis sur le bouton Options. Dans la fenêtre qui s'affiche, sélectionnez Compléments dans le volet gauche. La fenêtre représentée à la figure 2-15 s'affiche. Cliquez sur le bouton Atteindre à côté de la zone Gérer.

   Les compléments sont affichés. Les compléments actifs sont cochés.
2. Cliquez sur le bouton Parcourir. Si vous avez enregistré le complément dans le dossier proposé par défaut, il apparaît dans la liste. Sélectionnez le fichier voulu et validez.
3. Cochez ou décochez le complément. Le nom et le descriptif qui apparaissent dans la boîte de dialogue Compléments sont ceux qui ont été indiqués comme titre et commentaire du fichier (voir figure 2-16).

Les compléments intégrés d'Excel

Excel est livré avec des compléments, tels que le Solveur. Ceux-ci doivent être installés pour être accessibles. Si ce n'est pas le cas, vous devez relancer l'installation d'Office et installer les compléments non disponibles.

Conseil

Avant de vous lancer dans des activités de programmation complexes, vérifiez s'il n'existe pas un complément intégré répondant à vos besoins.

Définir le classeur de stockage lors de l'enregistrement d'une macro

Lors de l'enregistrement, la zone Enregistrer la macro dans permet de définir le classeur dans lequel sera stockée la macro. Trois options sont disponibles :

- Classeur de macros personnel. La macro sera enregistrée dans le classeur de macros personnel et accessible à partir de n'importe quel classeur.
- Nouveau classeur. Un nouveau classeur est créé et la macro y est stockée.
- Ce classeur. La macro est stockée dans le classeur actif. Il peut s'agir du classeur de macros personnel (PERSONAL.XLSB).

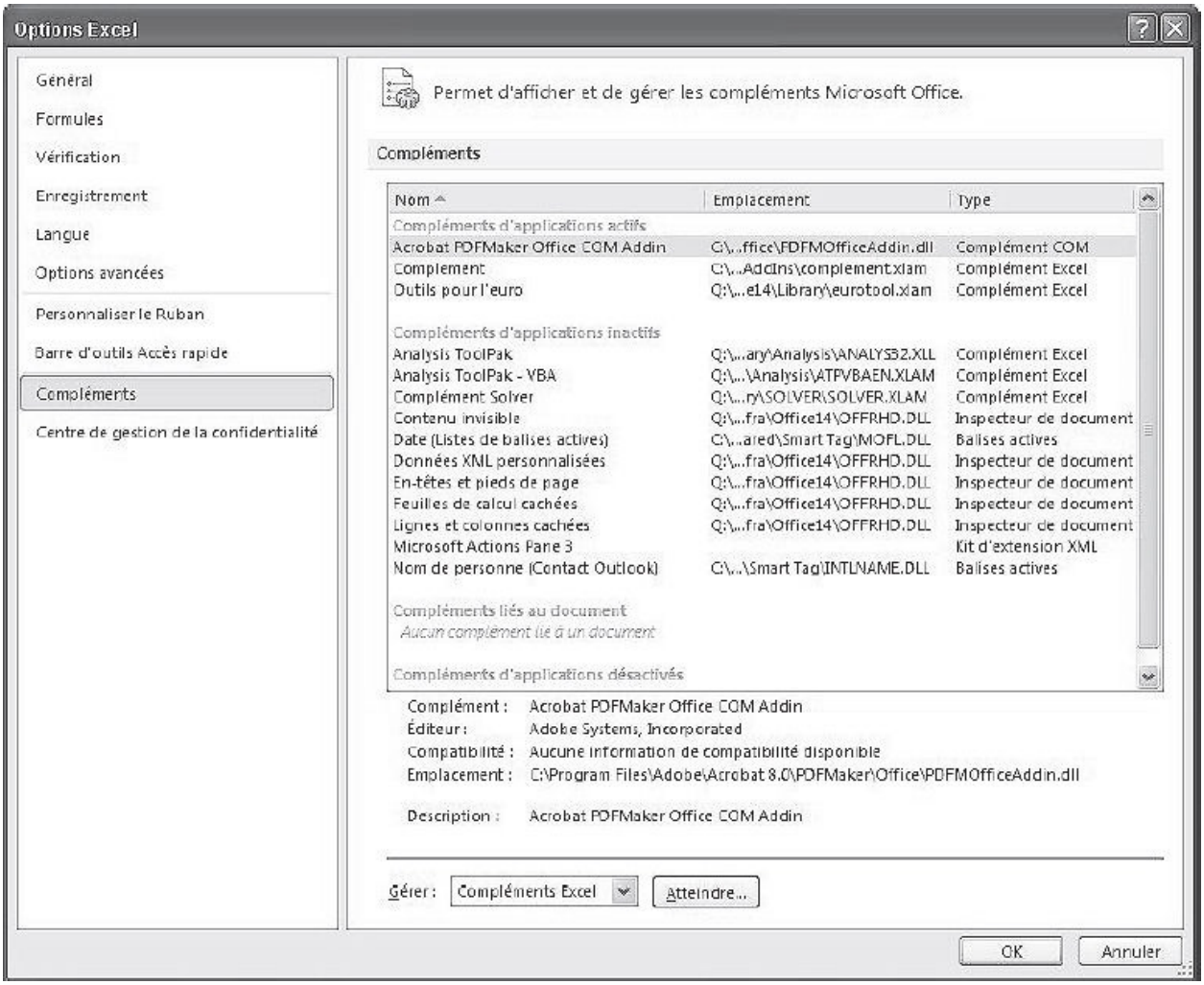

Options Excel
Général
Formules
Vérification
Enregistrement
Langue
Options avancées
Personnaliser le Ruban
Barre d'outils Accès rapide
Compléments
Centre de gestion de la confidentialité
Permet d'afficher et de gérer les compléments Microsoft Office.
Compléments
Nom
Emplacement
Type
Compléments d'applications actifs
Acrobat PDFMaker Office COM Addin C:\...ffice\PDFMOfficeAddin.dll Complément COM
Complement C:\...AddIns\complement.xlam Complément Excel
Outils pour l'euro C:\...e14\Library\eurotool.xlam Complément Excel
Compléments d'applications inactifs
Analysis ToolPak C:\...ary\Analysis\ANALYS32.XLL Complément Excel
Analysis ToolPak - VBA C:\...\Analysis\ATPVBAEN.XLAM Complément Excel
Complément Solver C:\...ry\SOLVER\SOLVER.XLAM Complément Excel
Contenu invisible C:\...fra\Office14\OFFRHD.DLL Inspecteur de document
Date (Listes de balises actives) C:\...ared\Smart Tag\MOFL.DLL Balises actives
Données XML personnalisées C:\...fra\Office14\OFFRHD.DLL Inspecteur de document
En-têtes et pieds de page C:\...fra\Office14\OFFRHD.DLL Inspecteur de document
Feuilles de calcul cachées C:\...fra\Office14\OFFRHD.DLL Inspecteur de document
Lignes et colonnes cachées C:\...fra\Office14\OFFRHD.DLL Inspecteur de document
Microsoft Actions Pane 3 Kit d'extension XML
Nom de personne (Contact Outlook) C:\...\Smart Tag\INTLNAME.DLL Balises actives
Compléments liés au document
Aucun complément lié à un document
Compléments d'applications désactivés
Complément : Acrobat PDFMaker Office COM Addin
Éditeur : Adobe Systems, Incorporated
Compatibilité : Aucune information de compatibilité disponible
Emplacement : C:\Program Files\Adobe\Acrobat 8.0\PDFMaker\Office\PDFMOfficeAddin.dll
Description : Acrobat PDFMaker Office COM Addin
Gérer :
Compléments Excel
Atteindre...
OK
Annuler

Figure 2-15 – *La liste des compléments d'Excel.*

Figure 2-16 – *Notre macro complémentaire apparaît maintenant dans la liste des macros complémentaires d'Excel.*

Figure 2-17 – *Définissez le classeur qui hébergera une macro lors de l'enregistrement de cette dernière.*

Accéder aux macros d'un classeur spécifique

Dans la liste déroulante Macros de la boîte de dialogue Macro, quatre options s'offrent à vous :

- Tous les classeurs ouverts. Les macros de tous les classeurs ouverts (y

compris le classeur de macros PERSONAL.XLSB) sont accessibles. Les macros qui ne sont pas stockées dans le classeur de macros personnel apparaissent sous la forme `Classeur`!`NomMacro`.

- Ce classeur. Seules les macros du classeur actif sont affichées. Elles apparaissent alors simplement sous la forme `NomMacro`.
- PERSONAL.XLSB. Seules les macros du classeur personnel sont accessibles.
- ***Nom_Classeur***. Vous pouvez aussi choisir de ne visualiser que les macros de l'un des classeurs ouverts, en sélectionnant simplement son nom dans la liste.

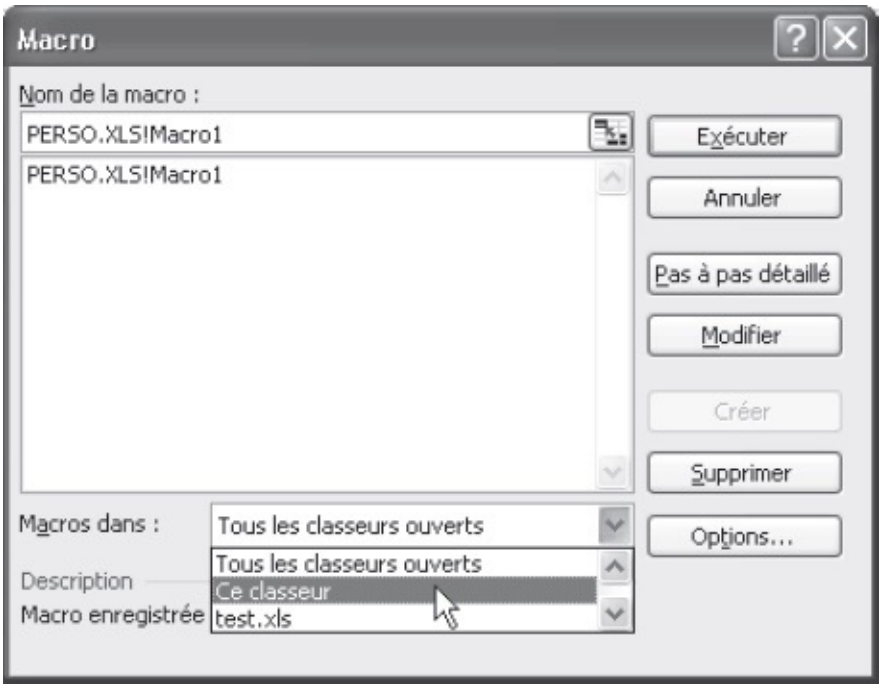


Figure 2-18 – *Vous pouvez définir le classeur dont vous souhaitez visualiser les macros.*

Attention

Si vous fermez le classeur personnel, vous ne pourrez plus accéder aux macros qui y sont stockées, ni y enregistrer de nouvelles macros. Le classeur personnel s'ouvrira de nouveau lors de la prochaine session Excel. Pour enregistrer des macros d'accès global ou accéder à celles du classeur personnel au cours de la session active, vous devez rouvrir PERSONAL.XLSB.

Astuce

Le classeur PERSONAL.XLSB s'ouvre à l'exécution d'Excel parce qu'il se trouve dans le dossier

XLSTART (ou XLOuvrir selon la version d’Office). Pour ouvrir automatiquement un classeur au lancement d’Excel, créez un raccourci vers ce classeur et placez-le dans ce dossier.

3

Déplacement et sélection dans une macro Excel

Le déplacement et la sélection de cellules dans un classeur sont primordiaux lors de l'enregistrement de macros. Il est indispensable d'en connaître les techniques et de comprendre les concepts de référence relative ou absolue aux cellules, pour créer des macros qui se comportent comme vous le souhaitez.

Pour enregistrer des déplacements dans une feuille Excel, vous pouvez utiliser indifféremment le clavier ou la souris. L'emplacement de la cellule active est enregistré lorsque vous effectuez une opération (mise en forme, saisie, etc.) qui modifie la feuille Excel. Autrement dit, si vous vous contentez de vous déplacer dans la feuille – par des clics de souris ou en utilisant les flèches du clavier – sans jamais intervenir sur le contenu ou la mise en forme de la cellule, ces déplacements ne seront pas enregistrés dans la macro.

La sélection d'éléments dans une feuille Excel repose sur l'objet `Range` qui représente une cellule, une ligne, une colonne ou une combinaison de ces éléments. Comme vous le verrez dans cette section, les propriétés utilisées varient selon le type de sélection effectué, mais toutes renvoient un objet `Range`.

Le codage en Visual Basic de vos déplacements varie selon que vous activez ou non la référence relative aux cellules, en cliquant sur le bouton correspondant de la barre d'outils Arrêt de l'enregistrement (voir figure 3-1) ou dans la zone Code de l'onglet Développeur si vous utilisez Excel 2007.

Figure 3-1 – *Vous pouvez enregistrer vos déplacements par référence relative*

ou absolue aux cellules.

Méthodes de sélection dans une feuille Excel

Clavier

L'enregistrement de déplacements dans une feuille Excel par référence relative aux cellules nécessite, dans certains cas, que vous utilisiez le clavier (par exemple, si vous souhaitez activer la dernière cellule non vide d'une ligne). Le tableau 3-1 présente les différentes possibilités.

Tableau 3-1. Déplacement dans une feuille Excel à l'aide du clavier

Pour se déplacer	Clavier
D'une cellule vers la droite	→
D'une cellule vers la gauche	←
D'une cellule vers le haut	↑
D'une cellule vers le bas	↓
Au début de la ligne courante	Début
Sur la première cellule non vide de la ligne courante[1]	Ctrl + ←
Sur la dernière cellule non vide de la ligne courante[1]	Ctrl + →
Sur la première cellule non vide de la colonne courante[1]	Ctrl + ↑
Sur la dernière cellule non vide de la colonne courante[1]	Ctrl + ↓
Sur la cellule située à l'angle supérieur gauche de la feuille active (A1)	Ctrl + Début
Sur la cellule située à l'angle inférieur droit de la feuille active[2]	Ctrl + Fin

1. Si des cellules vides se trouvent entre la cellule active et la cellule visée, Excel s'arrête successivement sur les cellules contiguës aux cellules vides. Par exemple, si la cellule B10 est active et si toutes les cellules situées au-dessus d'elle contiennent des données à l'exception de B5, la répétition de Ctrl + ↑ entraînera l'activation successive des cellules B6, puis B4 et enfin B1.
2. L'adresse de la cellule située à l'angle inférieur droit de la feuille active est la combinaison de la dernière colonne de la dernière ligne contenant des données. Cette cellule peut être vide.

Attention

Lors de l'enregistrement d'une macro par référence absolue aux cellules, c'est l'adresse des cellules qui est mémorisée. Pour un déplacement relatif (par exemple, la dernière cellule non vide de la ligne courante), vous devez activer l'enregistrement par référence relative aux cellules en cliquant sur le bouton correspondant de la barre d'outils Arrêt de l'enregistrement.

Pour étendre la sélection de la cellule active à une cellule donnée de la feuille, utilisez l'une des combinaisons de touches présentées dans le tableau 3-1, en maintenant la touche Maj enfoncée. Pour sélectionner les colonnes entières correspondant aux cellules sélectionnées, utilisez le raccourci clavier Ctrl+Barre d'espace ; pour sélectionner les lignes entières, utilisez Maj + Barre d'espace.

Notez que, pour sélectionner des zones non contiguës, vous devez obligatoirement utiliser la souris.

Souris

Pour sélectionner une cellule, cliquez dessus. Pour sélectionner une ligne ou une colonne, cliquez sur son en-tête.

Pour sélectionner des cellules adjacentes, cliquez sur la première cellule de la plage puis, tout en appuyant sur la touche Maj, cliquez sur la dernière cellule. Pour sélectionner des lignes ou des colonnes adjacentes, procédez de la même façon, en cliquant sur leurs en-têtes.

Pour sélectionner des cellules non contiguës, cliquez sur la première cellule puis, tout en appuyant sur la touche Ctrl, cliquez successivement sur les autres cellules. Pour sélectionner des lignes ou des colonnes non contiguës, procédez de la même façon, en cliquant sur leurs en-têtes.

Il est possible de combiner la sélection d'éléments non contigus de la feuille en maintenant la touche Ctrl enfoncée et en cliquant sur les éléments voulus. Vous pouvez, par exemple, sélectionner simultanément la colonne C, la ligne 5 et la cellule F4 : cliquez sur l'en-tête de la colonne C puis, tout en appuyant sur la touche Ctrl, cliquez sur l'en-tête de la ligne 5 et enfin sur la cellule F4.

Notion de cellule active

Lorsqu'une plage de cellules est sélectionnée dans une feuille Excel, toutes les cellules de cette plage, à l'exception d'une seule, sont noircies. Celle qui n'est pas noircie est la cellule active de la plage (voir figure 3-2).

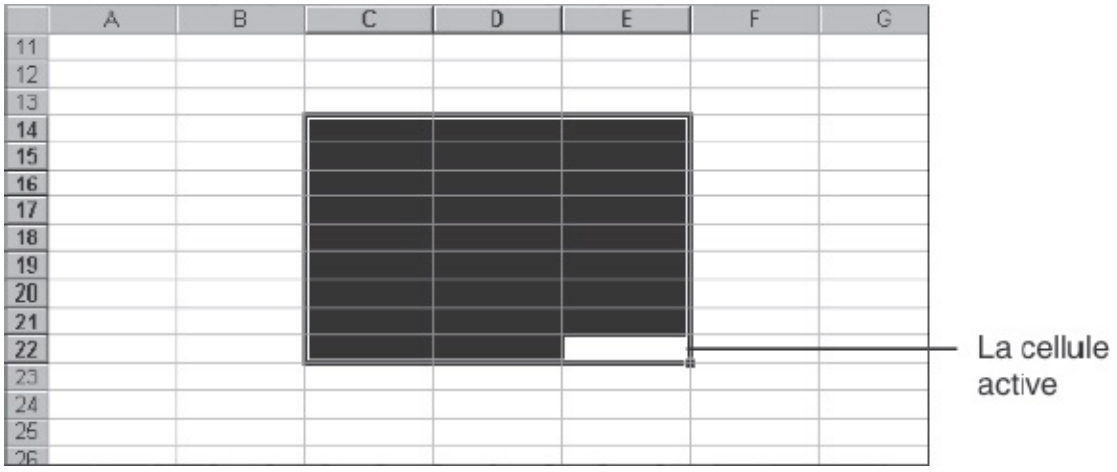


Figure 3-2 – *Dans une plage de cellules Excel, une seule est la cellule active.*

Si vous appliquez une mise en forme (une police particulière, par exemple), elle concernera l'ensemble des cellules de la plage sélectionnée. Si vous appuyez sur la touche Suppr, le contenu de toutes les cellules de la plage sera supprimé. Plus généralement, si vous effectuez une opération pouvant affecter simultanément plusieurs cellules, elle s'appliquera à l'ensemble de la plage sélectionnée.

Cependant, certaines opérations – comme la saisie de texte ou de formules – ne peuvent s'appliquer qu'à une cellule à la fois. C'est alors la cellule active qui est affectée. Par exemple, si la plage de cellules A5+F10 est sélectionnée et si A5 est la cellule active, le texte saisi au clavier sera inséré dans A5 et la plage A5+F10 restera sélectionnée.

Pour nommer une plage, on indique la cellule située à l'angle supérieur gauche (A5), puis celle située à l'angle inférieur droit (F10).

La cellule active d'une plage dépend de l'ordre dans lequel vous avez sélectionné les différentes cellules qui la composent. Vous verrez dans les sections qui suivent comment le mode de sélection détermine la cellule active et comment cette dernière est codée en Visual Basic.

Références relatives et références absolues

Par défaut, l'enregistrement s'effectue par référence absolue à des cellules. Cela signifie que, lorsque vous vous déplacez dans une feuille Excel, l'Enregistreur de macro mémorise l'adresse de la cellule de destination (combinaison du numéro de ligne et de la lettre de colonne). Ainsi, si vous enregistrez un déplacement vers B6, l'exécution de la macro entraînera

l'activation de cette dernière, quelle que soit la cellule active au moment du lancement de la macro.

Le bouton Référence relative de la barre d'outils Arrêt de l'enregistrement sert à enregistrer les déplacements dans la feuille Excel relativement à la cellule initialement active. Ce n'est plus l'adresse de la cellule qui est prise en considération, mais le déplacement dans la feuille. Ainsi, le passage de B5 à C7 sera enregistré comme un déplacement d'une colonne vers la droite et de deux lignes vers le bas. Si, au moment de l'exécution de la macro, la cellule active est D1, alors E3 (située une colonne à droite et deux lignes en dessous de D1) sera à son tour activée.

Lors de l'enregistrement de macros dans Excel, vous pouvez combiner les références relatives et absolues, en cliquant sur le bouton Référence relative chaque fois que vous voulez changer. Observez le classeur représenté à la figure 3-3. La colonne D contient les chiffres d'affaires effectués par les représentants. La colonne E doit contenir les primes.

	A	B	C	D	E
1	Représentant	Région	Ref	Chiffre du mois	Prime
4	Artuis J.	Ouest	0078	54 253	
5	Bertrand Isabelle	Nord-est	0072	89 653	
6	Bidault Jean	Est	0079	48 625	
7	Boitier Henri	Ouest	0008	78 545	
8	Cartois Hervé	Ouest	0077	102 565	
9	Denis Emmanuelle	Sud	0074	35 007	
10	Dupin Stéphane	Sud-oues	0073	69 854	
11	Frimousse Sylvie	Sud-est	0098	75 215	
12	Goraguer Vivienne	Centre	0001	59 303	
13	Gropois Yves	Nord	0097	82 353	
14	Jean Mickael	Centre	0128	49 953	
15	Le Bras Laetitia	Centre	0178	28 503	
16	Lemaire Francis	Nord-Ouest	0139	95 863	
17	Les Neury Yvonne	Ouest	0005	45 233	
18	Letendre Alain	Est	0145	98 503	
19	Limouse Evelyne	Sud-est	0153	68 503	

Figure 3-3 – *La combinaison des références relatives et absolues permettra de calculer les primes des représentants.*

Pour calculer la prime, il suffit de procéder comme suit :

1. Se placer dans la première cellule contenant un chiffre d'affaires (D4).
2. Calculer la prime – vous apprendrez au chapitres 5 à créer des fonctions personnalisées et à les exploiter dans vos programmes VBA.
3. Se déplacer d'une cellule vers la droite (E4).

4. Insérer le résultat issu du calcul.
5. Se déplacer d'une cellule vers le bas, puis d'une cellule vers la gauche, afin d'atteindre le chiffre suivant (D5).
6. Recommencer au point 2 si la cellule sélectionnée contient une valeur.

Si vous souhaitez créer un programme VBA prenant en charge ce calcul, il vous faudra utiliser une référence absolue aux cellules pour le point 1 et une référence relative pour les déplacements des points 3 et 5. Il suffit pour cela de vous assurer que l'enregistrement de la macro s'effectue par référence absolue avant de sélectionner D4, puis de cliquer sur le bouton Référence relative avant les deux autres étapes.

Attention

La référence aux cellules (relative ou absolue) active au moment où vous interrompez l'enregistrement d'une macro sera aussi la référence active si vous enregistrez une nouvelle macro dans la même session Excel (sans avoir quitté, puis relancé l'application). Lorsque vous enregistrez une macro, pensez toujours à vérifier la référence avant de commencer à vous déplacer dans la feuille.

Coder les déplacements effectués lors de l'enregistrement d'une macro

Les déplacements dans une feuille sont interprétés en Visual Basic comme la manipulation d'objets Excel. Il s'agit d'objets `Range` auxquels vous accéderez à l'aide des propriétés suivantes :

- **`Range`**. Renvoie un objet `Range` représentant une cellule, une plage de cellules, ou un groupe de cellules non contiguës.
- **`Cells`**. Renvoie la collection `Cells` qui représente toutes les cellules du classeur actif. Permet aussi de renvoyer une cellule ou une plage de cellules spécifiée.
- **`Row`**. Renvoie un numéro représentant une ligne de la feuille. La propriété `Rows` renvoie un objet `Range` qui représente toutes les lignes de la feuille ou toutes les lignes d'un objet `Range` spécifié.
- **`Column`**. Renvoie un numéro représentant une colonne de la feuille. La propriété `Columns` renvoie un objet `Range` qui représente toutes les colonnes de la feuille ou toutes les colonnes d'un objet `Range` spécifié.
- **`ActiveCell`**. Renvoie un objet `Range` représentant la cellule active d'une feuille de calcul.

- **`Selection`**. Renvoie l'objet sélectionné dans la feuille de calcul active : un objet `Range` représentant une cellule ou un groupe de cellules.
- **`Offset`**. Renvoie un objet `Range` qui représente une plage décalée par rapport à la plage spécifiée.

Vous apprendrez à exploiter ces objets à l'aide des méthodes suivantes :

- **`Select`** et **`GoTo`**. Sélectionnent l'objet spécifié.
- **`Activate`**. Active l'objet spécifié. S'il s'agit d'un `Range`, la cellule spécifiée devient la cellule active. Si un groupe de cellules est sélectionné, la sélection est maintenue.
- **`Resize`**. Modifie l'ampleur d'une plage de cellules.

Référence absolue aux cellules

Cette section présente le codage VBA des déplacements par référence absolue aux cellules.

Sélection de cellules contiguës

L'expression Visual Basic pour sélectionner une cellule par référence absolue se présente ainsi :

```
Range("Adresse_Cellule").Select
```

- La propriété `Range` indique qu'il s'agit d'un objet `Range`.
- L'argument *`Adresse_Cellule`* précise l'adresse de cet objet. Cet argument est composé de la référence de colonne (une lettre) immédiatement suivie de la référence de ligne (un nombre).
- La méthode `Select` entraîne la sélection de l'objet `Range` précédemment défini (ici la cellule).

L'instruction Visual Basic pour activer la cellule B5 de la feuille active est la suivante :

```
Range("B5").Select
```

Pour sélectionner une cellule ou une plage, la feuille doit être active. Si tel n'est pas le cas, commencez par activer la feuille voulue à l'aide de la méthode `Activate`. Les instructions permettant de sélectionner la cellule B5 de la feuille intitulée Janvier du classeur Ventes.xlsx – sans qu'il soit nécessaire que Janvier soit la feuille active – sont les suivantes :

```
Workbooks("Ventes.xlsx").Sheets("Janvier").Activate
Range("B5").Select
```

Info

Nous considérerons dans la suite de ce chapitre que la sélection s'effectue sur la feuille active et omettrons donc toute instruction d'activation.

Astuce

La propriété `Cells` permet aussi de coder une référence absolue à une cellule. Elle s'utilise avec la syntaxe suivante :

`Cells(`*`ligne, colonne`*`)`

où *`ligne`* est l'index de ligne et *`colonne`* l'index de colonne, tous deux exprimés par un chiffre – l'argument colonne prend la valeur 1 pour la colonne A, 2 pour la colonne B, etc. Les expressions `Cells(2,5).Select` et `Range("E2").Select` sont donc strictement équivalentes. Lorsque vous enregistrez un déplacement par référence absolue aux cellules, il est toujours codé à l'aide de la propriété `Range`. La propriété `Cells` offre l'avantage de pouvoir faire référence à des cellules à l'aide de variables numériques – les variables sont traitées au chapitres 6.

Lorsque vous sélectionnez une plage de cellules, le code Visual Basic généré se présente ainsi :

```
Range("Cell1:Cell2").Select
Range("Cell_Active").Activate
```

où *`Cell1`* et *`Cell2`* représentent respectivement la cellule située à l'angle supérieur gauche de la plage et celle située à l'angle inférieur droit.

Dans l'expression `Range("`*`Cell_Active`*`").Activate`, l'argument *`Cell_Active`* indique la cellule active dans la plage sélectionnée. Dans Excel, cette cellule qui n'est pas noircie est celle à partir de laquelle la sélection a été étendue. Si vous saisissez du texte au clavier, il sera inséré dans cette cellule.

Ainsi, l'expression Visual Basic :

```
Range("B5:D10").Select
Range("D5").Activate
```

revient à sélectionner une plage dont les cellules situées aux angles supérieur gauche et inférieur droit sont respectivement B5 et D10. Cependant, cette sélection a été effectuée en partant de D5 et en étendant la plage jusqu'à B10, si bien que la cellule active de la sélection est D5 (voir figure 3-4).

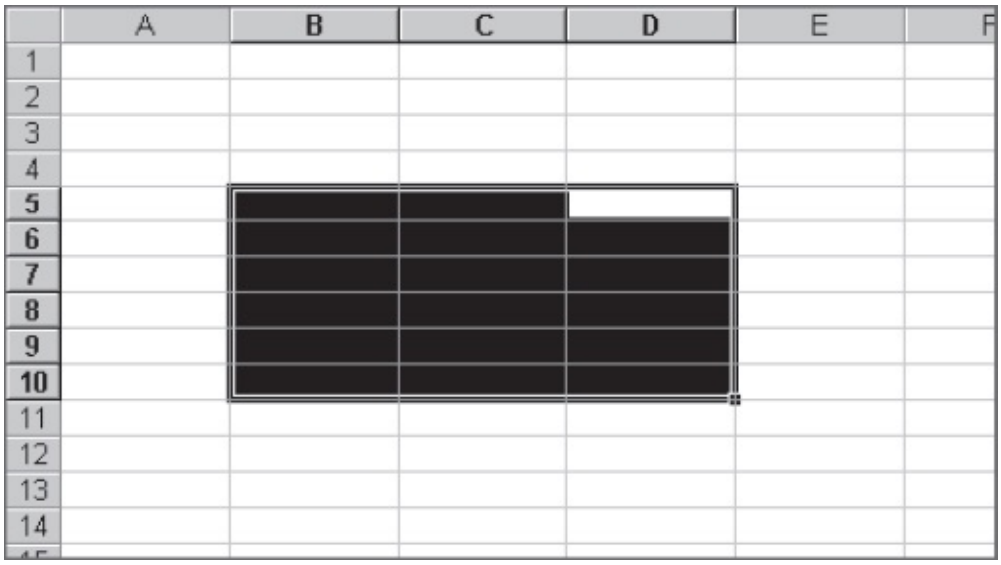


Figure 3-4 – *La cellule active est D5.*

Sélection de cellules non contiguës

Pour sélectionner des cellules non contiguës dans une feuille Excel, il faut maintenir la touche Ctrl enfoncée. La syntaxe de la propriété `Range` lors de la sélection de cellules non contiguës est la suivante :

```
Range("Cell1,Cell2,…,Celln").Select
Range("Cell_Active").Activate
```

où les arguments *`Cell1`* à *`Celln`* représentent les cellules successivement sélectionnées. Lors de l'enregistrement d'une macro, l'argument *`Cell_Active`* représente la dernière cellule sélectionnée.

Par exemple, l'expression Visual Basic suivante :

```
Range("B5, D10, F2, A3").Select
Range("A3").Activate
```

revient à sélectionner successivement les cellules B5, D10, F2 et A3 en maintenant la touche Ctrl enfoncée.

Sélection de lignes et de colonnes

Lorsque vous sélectionnez une colonne dans une feuille Excel, l'Enregistreur de macro code cette sélection à l'aide de la propriété `Columns`. Dans le cas d'une ligne, c'est `Rows` qui sera utilisée. Ces propriétés renvoient toutes deux des objets `Range`, de type colonne pour `Columns` et de type ligne pour `Rows`.

Rappel

Pour sélectionner une colonne ou une ligne dans une feuille Excel, cliquez sur son en-tête ou utilisez l'un des raccourcis clavier présentés dans le tableau 3-1. Pour des colonnes ou des lignes contiguës, sélectionnez la première ligne/colonne, puis enfoncez la touche Maj et cliquez sur la dernière ligne/colonne de la plage.

Lignes contiguës

La syntaxe de la propriété `Rows` est la suivante :

```
Rows("ligne1:ligne2").Select
Range("cell_active").Activate
```

où les arguments *`ligne1`* et *`ligne2`* représentent respectivement les index de la première et de la dernière ligne de la plage. La méthode `Select` sélectionne l'objet `Range` défini par la propriété `Rows`.

Notez que l'expression `Range("cell_active").Activate` est omise si la cellule active est celle située à l'angle supérieur gauche de la plage sélectionnée.

Info

Lorsque vous définissez une plage de lignes, la cellule active est la première cellule de la première ligne que vous sélectionnez. Par exemple, si, lors de l'enregistrement d'une macro, vous sélectionnez la ligne 5, puis maintenez la touche Maj enfoncée et sélectionnez la ligne 10, c'est la cellule A5 (située à l'angle supérieur gauche de la plage) qui sera active. En revanche, si vous sélectionnez la ligne 10, puis la 5, c'est la cellule A10 qui sera active.

Si la sélection ne porte que sur une ligne, les arguments *`ligne1`* et *`ligne2`* ont la même valeur et l'expression `Range("cell_active").Activate` est omise. Par exemple, si vous enregistrez dans une macro la sélection de la ligne 5 de la feuille active, le code Visual Basic correspondant se présentera ainsi :

```
Rows("5:5").Select
```

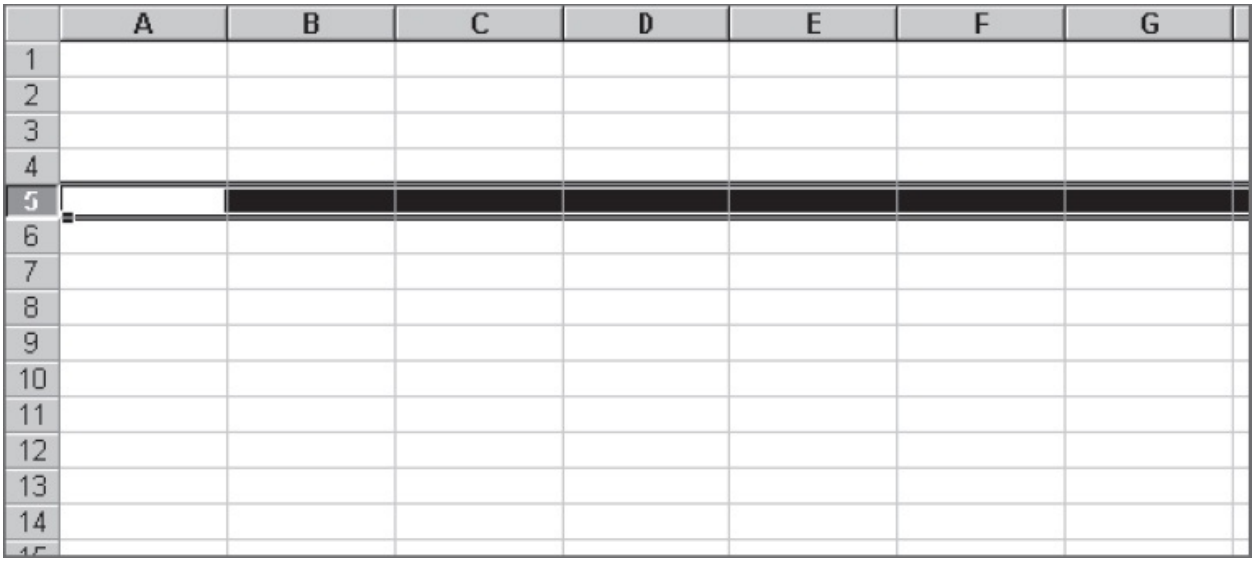


Figure 3-5 – *Lorsqu'une seule ligne est sélectionnée, le code ne spécifie pas de cellule active.*

Si vous sélectionnez les lignes 5 à 10 en commençant par la 5, le code Visual Basic correspondant se présentera ainsi :

```
Rows("5:10").Select
```

Si vous sélectionnez la même plage, mais en commençant par la ligne 10, le code Visual Basic correspondant se présentera ainsi :

```
Rows("5:10").Select
Range("A10").Activate
```

Lorsque vous sélectionnez une ligne, la cellule active est par défaut la première de la ligne. Si vous modifiez la cellule active – dans ou hors de la plage sélectionnée – en maintenant la touche Ctrl enfoncée et en cliquant sur la cellule que vous souhaitez activer, la propriété `Range` se substitue à la propriété `Rows`. Votre code se présente alors ainsi :

```
Range("ligne1:ligne2, cell_active").Select
Range("cell_active").Activate
```

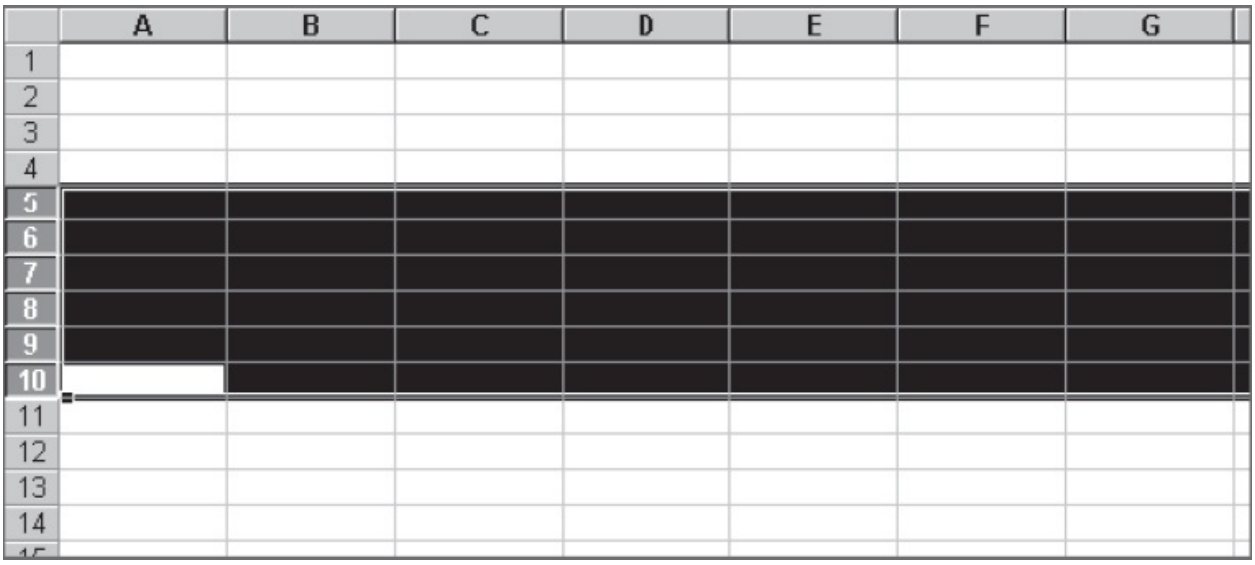


Figure 3-6 – *La cellule active est A10.*

où l'argument `cell_active` représente l'adresse de la cellule active. Par exemple, si vous sélectionnez la ligne 14, puis maintenez la touche enfoncée et cliquez sur la cellule B14 lors de l'enregistrement d'une macro, le code Visual Basic correspondant se présentera ainsi :

```
Range("14:14, B14").Select
Range("B14").Activate
```

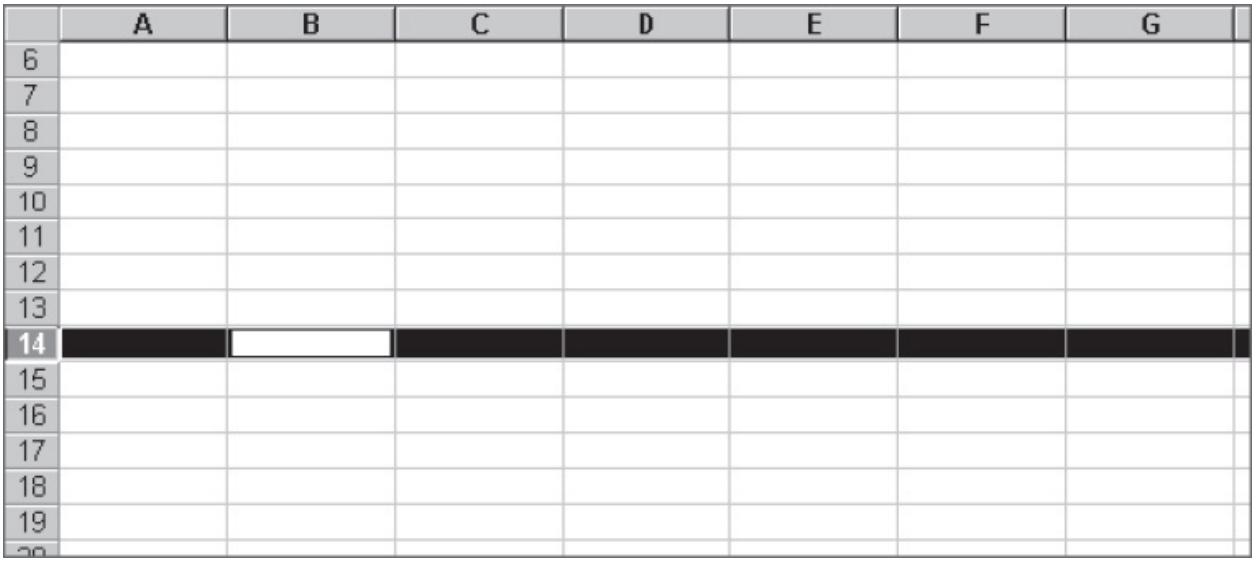


Figure 3-7 – *La cellule B14 est la cellule active.*

Colonnes contiguës

La syntaxe de la propriété `Columns` est la même que celle de `Rows` :

```
Columns("col1:col2").Select
Range("cell_active").Activate
```

Dans l'expression `Range("cell_active").Activate`, *cell_active* représente l'adresse de la cellule active dans la plage sélectionnée. Cette expression est omise si la cellule active est celle située à l'angle supérieur gauche de la plage.

Si la sélection ne porte que sur une colonne, les arguments *col1* et *col2* ont la même valeur et l'expression `Range("cell_active").Activate` est omise. Ci-dessous est présenté le code généré lors de l'enregistrement d'une macro consistant à sélectionner la colonne B de la feuille active :

```
Columns("B:B").Select
```

Si vous sélectionnez les colonnes B à E en commençant par B, le code Visual Basic correspondant se présentera ainsi :

```
Columns("B:E").Select
```

Si vous sélectionnez la même plage, mais en commençant par E, le code Visual Basic correspondant se présentera ainsi :

```
Columns("B:E").Select
Range("E1").Activate
```

Lorsque vous sélectionnez une colonne, la cellule active est par défaut la première de la colonne. Si vous modifiez la cellule active – dans ou hors de la plage sélectionnée – en maintenant la touche Ctrl enfoncée et en cliquant sur la cellule que vous souhaitez activer, la propriété `Range` se substitue à la propriété `Columns`. Votre code se présente alors ainsi :

```
Range("col1:col2, cell_active").Select
Range("cell_active").Activate
```

où l'argument *cell_active* représente l'adresse de la cellule active. Par exemple, si vous sélectionnez la colonne E, puis maintenez la touche Ctrl enfoncée et cliquez sur la cellule E5 lors de l'enregistrement d'une macro, le code Visual Basic correspondant se présentera ainsi :

```
Range("E:E, E5").Select
Range("E5").Activate
```

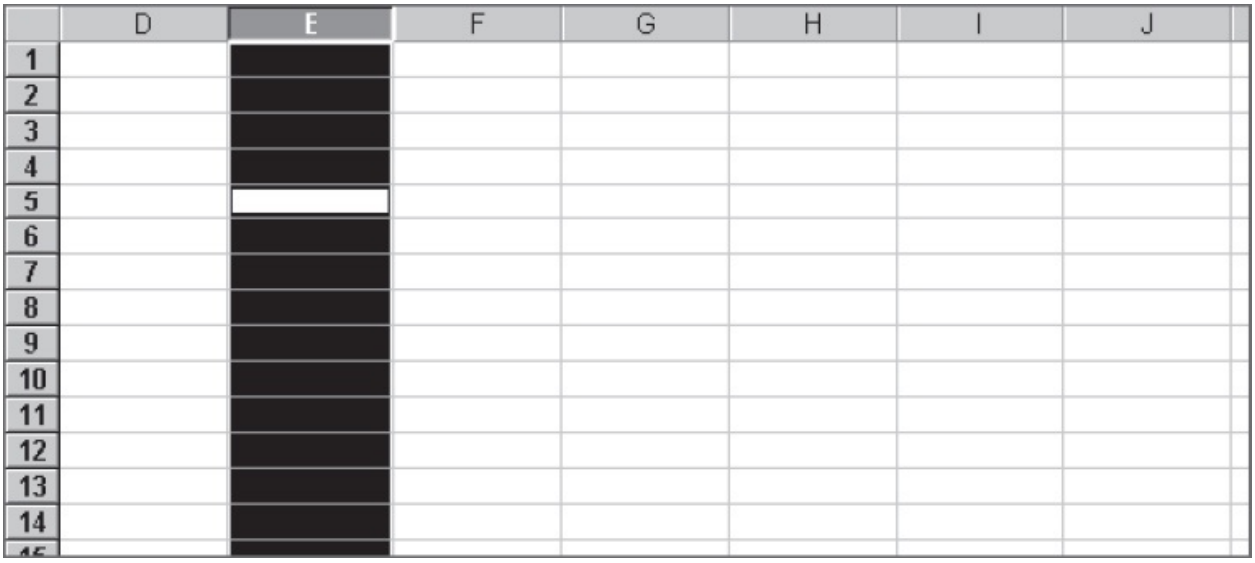


Figure 3-8 – *La cellule active est E5.*

Info

Vous pouvez substituer la propriété `Range` à `Rows` et `Columns` dans le code de votre macro, en conservant les mêmes arguments. Par exemple, les expressions `Range("5:10").Select` et `Rows("5:10").Select` correspondent toutes deux à la sélection des lignes 5 à 10 de la feuille active. Lors de l'enregistrement de macros, les sélections sont codées différemment de façon à faciliter la lecture du code.

Lignes et colonnes non contiguës

Lorsque vous sélectionnez des lignes ou des colonnes non contiguës lors de l'enregistrement d'une macro – en maintenant la touche Ctrl enfoncée –, la propriété `Range` est utilisée. La syntaxe se présente alors ainsi :

```
Range("item1:item2,item3:item4,…,item-n:item-n+1").Select
Range("cell_active").Activate
```

Les arguments *item* représentent les index des lignes ou colonnes sélectionnées. Ces arguments vont par paires, chaque paire représentant une plage de lignes ou de colonnes contiguës sélectionnées – le signe : est utilisé comme séparateur. Les arguments *item* d'une paire peuvent avoir une même valeur.

Si, par exemple, vous sélectionnez les colonnes A, C à E et G et si G1 est la cellule active, la syntaxe Visual Basic représentant cette sélection se présentera ainsi :

```
Range("A:A,C:E,G:G").Select
Range("G1").Activate
```

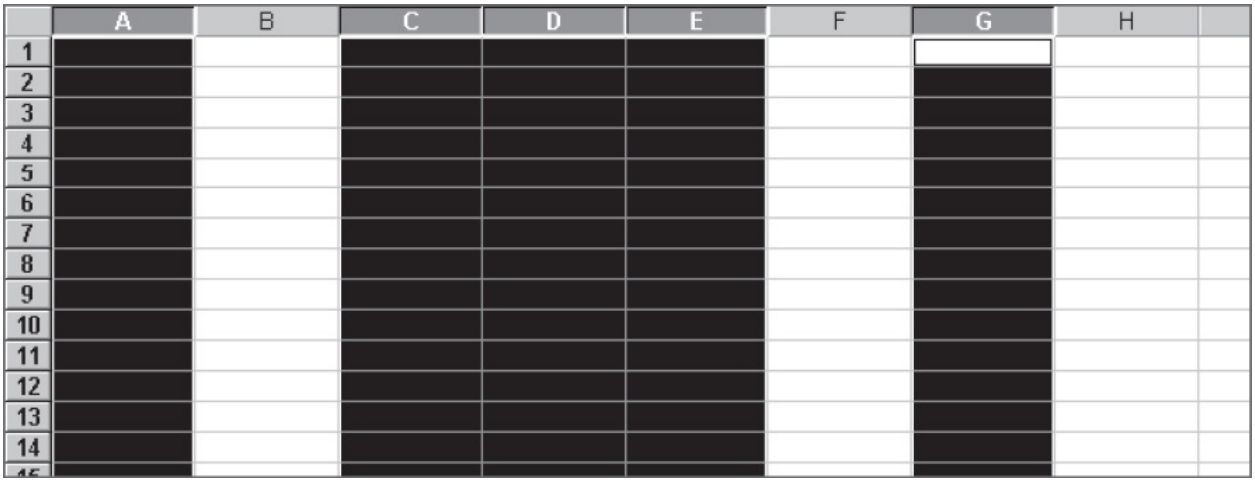


Figure 3-9 – *Vous pouvez conjuguer la sélection de colonnes adjacentes et de colonnes non contiguës.*

Info

Si, lors de l'enregistrement d'une macro, vous sélectionnez successivement des lignes ou des colonnes contiguës en maintenant la touche Ctrl enfoncée, plutôt que d'utiliser la touche Maj, ces sélections seront considérées comme autonomes et codées comme si les lignes ou les colonnes n'étaient pas contiguës.

Par exemple, si vous cliquez sur l'en-tête de la colonne A puis, tout en maintenant la touche Ctrl enfoncée, cliquez successivement sur les en-têtes des colonnes B, C et E, le code Visual Basic correspondant se présentera ainsi :

```
Range("A:A,B:B,C:C,E:E").Select
Range("C1").Activate
```

Si vous effectuez la même sélection en utilisant la touche Maj, le code correspondant se présente alors ainsi :

```
Range("A:C,E:E").Select
Range("C1").Activate
```

Vous pouvez, en utilisant les mêmes méthodes de sélection, définir une plage composée de lignes et de colonnes, contiguës ou non. Par exemple, si vous sélectionnez (à l'aide des touches Maj et Ctrl) les colonnes C à E, la colonne G, les lignes 4 à 6 et la ligne 8 comme indiqué à la figure 3-10, le code de votre macro se présentera ainsi :

```
Range("C:E,G:G,4:6,8:8").Select
Range("A8").Activate
```

Enfin, vous pouvez conjuguer la sélection de lignes, colonnes et cellules contiguës ou non. Le code suivant indique la même sélection, à laquelle on a ajouté les cellules B13 et A2 à F2.

```
Range("C:E,G:G,4:6,8:8,A2:F2,B13").Select
Range("B13").Activate
```

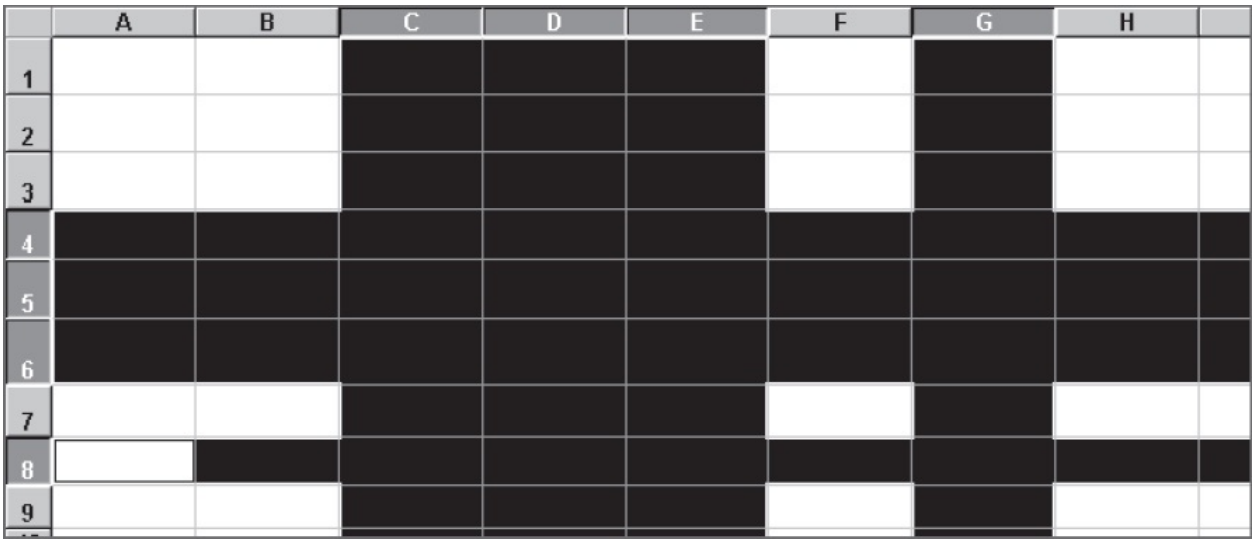


Figure 3-10 – *Vous pouvez conjuguer la sélection de colonnes et de lignes, contiguës ou non.*

Référence relative aux cellules

Le codage des déplacements par référence relative aux cellules répond aux mêmes principes qu'avec les références absolues. Cette section en présente les spécificités de façon sommaire.

Sélection de cellules contiguës

L'expression Visual Basic pour une référence relative à une cellule se présente ainsi :

```
ActiveCell.Offset(RowOffset, ColumnOffset).Range("A1").Select
```

- La propriété `ActiveCell` renvoie un objet `Range` qui représente la cellule active.
- La propriété `Offset` renvoie un objet `Range` (cellule ou plage de cellules), qui est fonction des arguments nommés *`RowOffset`* et *`ColumnOffset`*. Ceux-ci indiquent (en nombre de lignes et de colonnes) le décalage à effectuer à partir de la cellule active pour atteindre l'adresse de cet objet.

  Ces arguments prennent une valeur numérique – négative si le déplacement s'effectue vers le haut (*`RowOffset`*) ou vers la gauche (*`ColumnOffset`*).
- La propriété `Range("A1")` précise qu'il s'agit d'un déplacement de style A1. Autrement dit, la cellule active sert de référence ; elle est virtuellement considérée comme la cellule A1. Bien que toujours précisée lors de l'enregistrement d'une macro, cette propriété est facultative, et vous pouvez

la supprimer du code de la macro.

- La méthode `Select` entraîne la sélection de l'objet (ici la cellule) précédemment défini.

Par exemple, l'expression `ActiveCell.Offset(3,2).Select` entraînera la sélection de la cellule située trois lignes au-dessous et deux colonnes à droite de la cellule active.

L'expression `ActiveCell.Offset(-3,-2).Select` entraînera la sélection de la cellule située trois lignes au-dessus et deux colonnes à gauche de la cellule active.

Lorsque vous sélectionnez une plage de cellules, le code Visual Basic généré se présente ainsi :

```
ActiveCell.Offset(RowOffset, ColumnOffset).Range("A1:Cell2").Select
```

où `Offset(RowOffset, ColumnOffset)` indique le déplacement à effectuer relativement à la cellule active et `A1:Cell2` représente la plage de cellules sélectionnées en estimant que la cellule active est A1.

Par exemple, l'expression Visual Basic suivante :

```
ActiveCell.Offset(-3,0).Range("A1:A4").Select
```

revient à étendre la sélection de la cellule active jusqu'à celle située trois lignes au-dessus. On obtient alors une sélection équivalente à A1:A4 (il peut s'agir de B1:B4, C5:C8, etc.)

Sélection de cellules non contiguës

Vous pouvez sélectionner des cellules non contiguës dans une feuille en maintenant la touche Ctrl enfoncée. La syntaxe de la propriété `Range` est alors la suivante :

```
ActiveCell.Range("A1, Cell2,…, Celln").Select
```

où A1 représente la cellule active et les arguments *Cell2* à *Celln* sont les adresses des cellules successivement sélectionnées, par position relative à la cellule initiale. L'argument *Cell_Active* représente la dernière cellule sélectionnée.

> **Info**
> Lors de déplacements par référence relative aux cellules, le codage Visual Basic se fait par style de référence A1. Cela signifie que l'adresse virtuelle A1 est toujours attribuée à la cellule active, ce qui permet de représenter les déplacements et sélections d'autres cellules par rapport à cette adresse virtuelle.

Par exemple, l'expression suivante :

```
ActiveCell.Range("A1,A7,C7,C1").Select
ActiveCell.Offset(0,2).Range("A1").Activate
```

indique que la macro sélectionnera simultanément la cellule active (virtuellement A1) et celles dont les adresses virtuelles (en style de référence A1) sont A7, C7 et C1. La cellule active (virtuellement C1) sera celle située deux colonnes à droite – `Offset(0,2)` – de la cellule initialement active.

Si la cellule active au moment de l'exécution de la macro est B5, les cellules B5, B11, D11 et D5 seront sélectionnées. D5 sera la cellule active.

Pour bien comprendre ce principe, gardez à l'esprit que ce qui fait la relativité du déplacement est le repère d'origine. Les adresses des cellules sont déterminées par l'adresse de celle servant de repère.

Redimensionner une plage

Pour modifier l'ampleur d'une plage de cellules, utilisez la méthode `Resize` selon la syntaxe suivante :

```
expression.Resize(RowSize, ColumnSize)
```

où *`expression`* renvoie l'objet `Range` à redimensionner. `RowSize` et `ColumnSize`, facultatifs, sont respectivement les nombres de lignes et de colonnes qui doivent être ajoutées (valeurs positives) ou retirées (valeurs négatives) à la sélection. Si l'un ou l'autre de ces arguments est omis, il prend la valeur par défaut de 0 et la dimension correspondante n'est pas modifiée.

L'instruction suivante étend la sélection en cours d'une ligne et d'une colonne supplémentaires. Elle utilise pour cela la propriété `Count` qui, appliquée aux collections `Rows` et `Columns`, renvoie respectivement les nombres de lignes et de colonnes.

```
Selection.Resize(Selection.Rows.Count + 1, Selection.Columns.Count + 1).Select
```

Référence aux cellules en fonction de leur contenu

Vous pouvez effectuer ou modifier une sélection en fonction du contenu des cellules. Il sera ainsi utile d'étendre la sélection à l'ensemble des cellules non vides d'un tableau avant d'effectuer un tri ou d'identifier la première cellule vide dans une colonne afin d'y insérer du contenu. VBA propose pour cela les propriétés suivantes :

- **`CurrentRegion`.** Retourne la plage de cellules courante (les cellules contiguës qui contiennent des données).

- **`End`**. Permet d'identifier les cellules délimitant une plage de cellules.
- **`UsedRange`**. Retourne une plage composée des cellules d'une feuille contenant des données.

La propriété CurrentRegion

Pour étendre la sélection à la zone courante, c'est-à-dire à la zone entourée par une combinaison de lignes et de colonnes vides, vous utiliserez la propriété `CurrentRegion` selon la syntaxe suivante :

```
Range.CurrentRegion.Select
```

où *Range* est une expression qui renvoie un objet `Range`.

Considérez les deux instructions suivantes :

```
ActiveCell.CurrentRegion.Select
Range("C8").CurrentRegion.Select
```

La première instruction étend la sélection à la zone courante à partir de la cellule active, tandis que la seconde procède de même, mais à partir de la cellule C8 (figure 3-11).

	A	B	C	D	E	F	G	H	I	J	K
1			**traffic figures from 26th february to 5th march 2002**								
2											
3											
4				**DE**	**FR**	**UK**	**ES**	**IT**			
5			**26-févr**	22 480	52 634	5 922	98 498	52 634			
6			**27-févr**	34 423	46 889	4 533	45 867	46 889			
7			**28-févr**	32 279	40 978	4 188	27 678	40 978			
8			**01-mars**	29 628	51 158	3 267	87 649	51 158			
9			**02-mars**	30 877	33 545	5 168	68 795	33 545			
10			**03-mars**	26 647	31 143	5 056	57 869	31 143			
11			**04-mars**	22 500	34 837	5 777	40 985	34 837			
12			**05-mars**	25 800	36 885		36 885	36 885			
13											
14			**Total**	224 634	328 069	33 911					
15											
16			**average per day**	28 079	41 009	4 844					
17			**average per month**	**842 378**	**1 230 259**	**145 333**					
18											
19											
20			**Traffic increase evaluation**								
21											
22			30% more with portal integration	*1 120 362*	*1 636 244*	*193 293*					
23											
24											
25			20% more for new countries and new layout	*1 344 434*	*1 963 493*	*231 951*					
26											
27											
28			10% more directories and search engines promotion	*1 478 878*	*2 159 842*	*255 146*					

Figure 3-11 – *La propriété CurrentRegion permet de « capturer » une zone contenant des données à partir d'une cellule.*

La propriété End

La propriété `End` renvoie un objet `Range` qui représente la dernière cellule d'une zone. Cela revient à employer dans un tableau Excel la combinaison clavier Fin+flèche de direction. Utilisez la propriété `End` selon la syntaxe suivante :

```
Range.End(Direction)
```

où `Range` renvoie l'objet `Range` à partir duquel on recherche la dernière cellule non vide et où *`Direction`* représente le sens dans lequel on se déplace. Il peut s'agir de l'une des constantes `xlDirection` suivantes :

- **`xlDown`.** Déplacement vers le bas ;
- **`xlToRight`.** Déplacement vers la droite ;
- **`xlToLeft`.** Déplacement vers la gauche ;
- **`xlUp`.** Déplacement vers le haut.

Appliquées au tableau de la figure 3-11, les quatre instructions suivantes renvoient respectivement l'objet `Range` représentant les cellules E4, E12, C8 et H8.

```
Range("E8").End(xlUp).select         'renvoie la cellule E4
Range("E8").End(xlDown).select       'renvoie la cellule E12
Range("E8").End(xlToLeft).select     'renvoie la cellule C8
Range("E8").End(xlToRight).select    'renvoie la cellule H8
```

Vous pouvez évidemment utiliser la propriété `End` pour sélectionner des plages de cellules, comme nous l'avons vu précédemment dans ce chapitre. Considérez les exemples suivants.

- Sélection d'une plage de la première à la dernière cellule non vide d'une colonne :

```
Range("A1", Range("A1").End(xlDown)).Select
Range(ActiveCell, ActiveCell.End(xlDown)).Select
```

- Sélection d'une plage de la dernière cellule non vide jusqu'à la première :

```
Range("A32", Range("A32").End(xlUp)).Select
Range(ActiveCell, ActiveCell.End(xlUp)).Select
```

Si vous souhaitez sélectionner la première cellule vide d'une zone plutôt que la dernière cellule non vide, utilisez la propriété `Offset` pour décaler la sélection. La première instruction, ci-après, sélectionne la première cellule vide au bas de la colonne, tandis que la seconde sélectionne la première cellule vide à droite :

```
Range("A1").End(xlDown).Offset(1,0).Select
Range("A1").End(xltoRight).Offset(0,1).Select
```

La propriété UsedRange

La propriété `UsedRange` retourne la plage de cellules contenant les données d'une feuille. Cette propriété est donc particulièrement pratique pour identifier les cellules sur lesquelles doivent s'appliquer des traitements lors de l'exécution d'un programme. La plage retournée par `UsedRange` est un ensemble de cellules contiguës dont les limites sont définies par :

- la cellule dont l'index de colonne est le plus élevé ;
- la cellule dont l'index de colonne est le plus faible ;
- la cellule dont l'index de ligne est le plus élevé ;
- la cellule dont l'index de ligne est le plus faible.

L'instruction suivante sélectionne la plage de cellules utilisée sur la feuille active du classeur actif :

```
ActiveWorkbook.ActiveSheet.UsedRange.Select
```

Référence aux plages de cellules nommées

Pour faire référence à une plage de cellules nommée dans Excel, utilisez la syntaxe suivante :

```
Range("[NomClasseur]NomFeuille!NomPlage")
```

L'exemple suivant passe en gras les cellules de la plage nommée MaPlage, située sur la feuille Feuil1 du classeur Test.xlsx :

```
Range("[Test.xlsx]Feuil1!MaPlage").Font.Bold = True
```

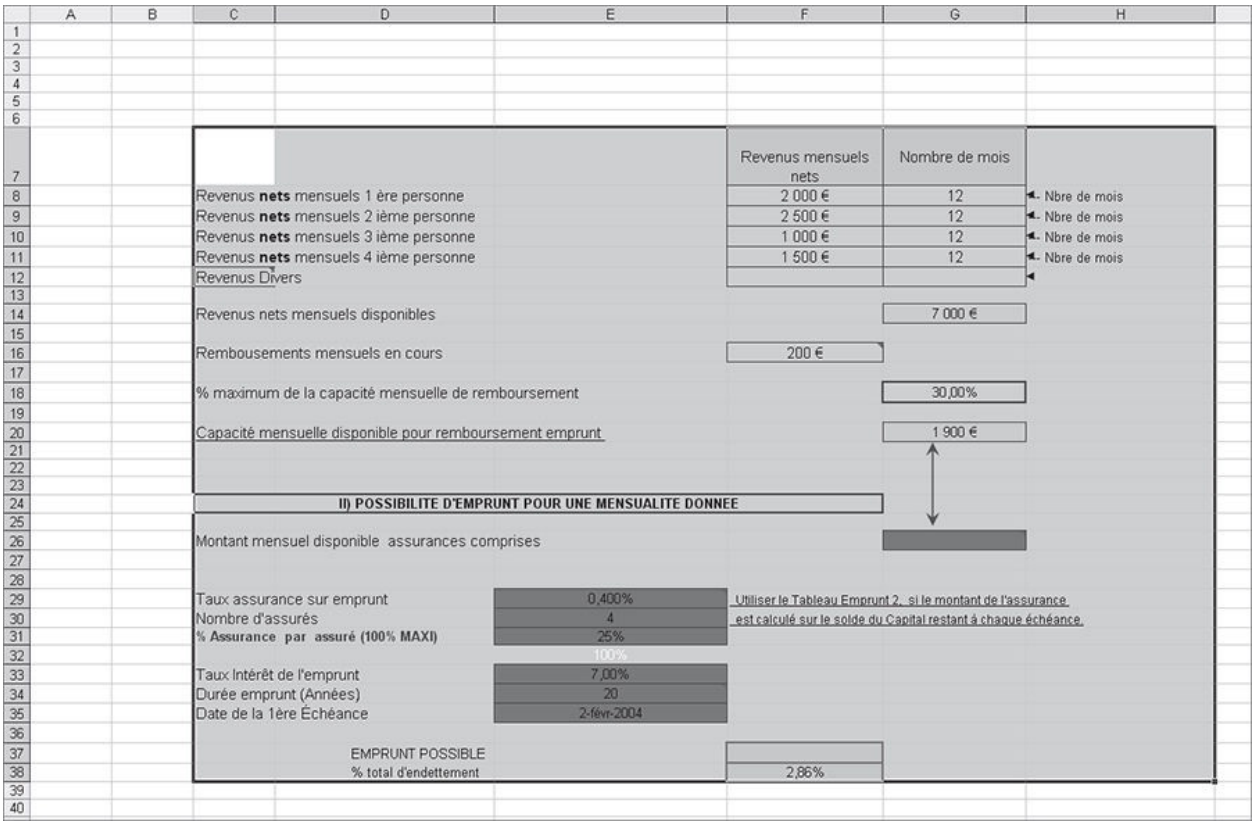


Figure 3-12 – *La propriété UsedRange retourne une plage qui englobe toutes les cellules contenant des données.*

Pour sélectionner une plage nommée, utilisez la méthode `GoTo` qui active successivement le classeur et la feuille si nécessaire, puis sélectionne la plage voulue. Les deux instructions suivantes sélectionnent la plage nommée MaPlage, puis en effacent le contenu :

```
Application.Goto Reference:="[Test.xlsx]Feuil1!MaPlage"
Selection.ClearContents
```

4

Découvrir Visual Basic Editor

Visual Basic Editor est l'environnement de développement intégré de VBA. C'est dans cet environnement que vous passerez l'essentiel de votre temps lors du développement de projets VBA. Les chapitres précédents vous ont fait découvrir la fenêtre Code à travers la modification et la création de macros. Toutefois, Visual Basic Editor ne se résume pas à un simple éditeur de code. Il s'agit d'un logiciel complet proposant des outils d'aide au développement, que ce chapitre vous propose de découvrir.

Accéder à Visual Basic Editor

Lorsque vous choisissez de modifier une macro existante ou d'en créer une nouvelle, selon les procédures étudiées au chapitres 2, vous accédez à la fenêtre Code de Visual Basic Editor. Vous pouvez aussi développer un projet VBA en accédant directement à Visual Basic Editor, sans passer par la boîte de dialogue Macro.

On accède toujours à l'environnement de développement à partir d'une application hôte. Autrement dit, une session Visual Basic Editor peut être liée à Word, PowerPoint ou encore Excel, mais ne permet d'accéder qu'aux projets de l'application à partir de laquelle il a été exécuté.

Info

Lorsque vous êtes dans Visual Basic Editor, vous pouvez accéder à l'ensemble des éléments constitutifs des projets accessibles, y compris aux macros disponibles dans la boîte de dialogue Macro.

Attention

Pour qu'un projet soit accessible dans Visual Basic Editor, il faut que le document dans lequel il est stocké soit ouvert dans l'application hôte.

Activez l'onglet Développeur du ruban, puis sélectionnez Visual Basic Editor,

ou tapez le raccourci clavier Alt+F11.

Rappel

L'onglet Développeur n'est pas activé par défaut. Pour l'afficher, rendez-vous sur la page Options et choisissez Personnaliser le ruban. Cochez la case Onglet développeur, puis validez.

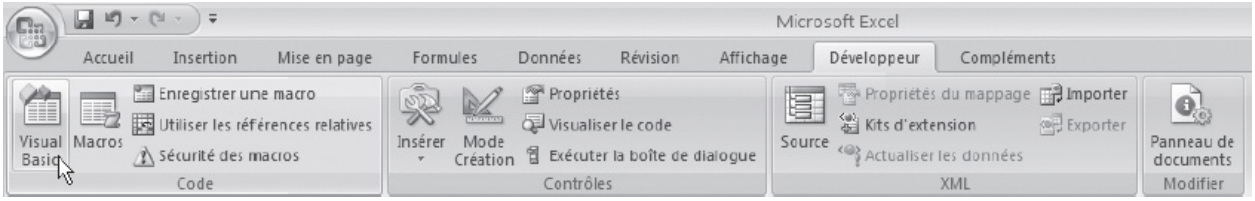


Figure 4-1 – *L'accès à Visual Basic Editor se fait via l'onglet Développeur.*

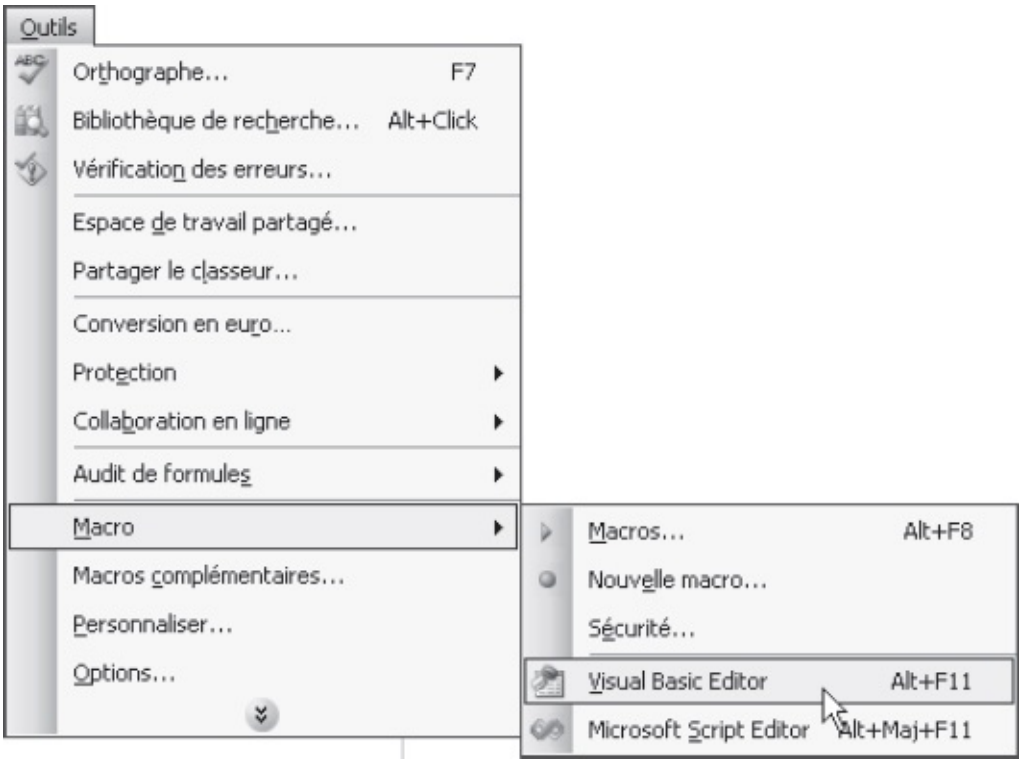


Figure 4-2 – *Dans les versions d'Excel antérieures à 2007, l'accès à Visual Basic Editor s'effectuait via le menu Outils !*

La figure 4-3 présente la fenêtre de Visual Basic Editor. Il se peut que, sur votre ordinateur, elle ne propose pas les mêmes éléments. Vous verrez par la suite comment en afficher les différents composants.

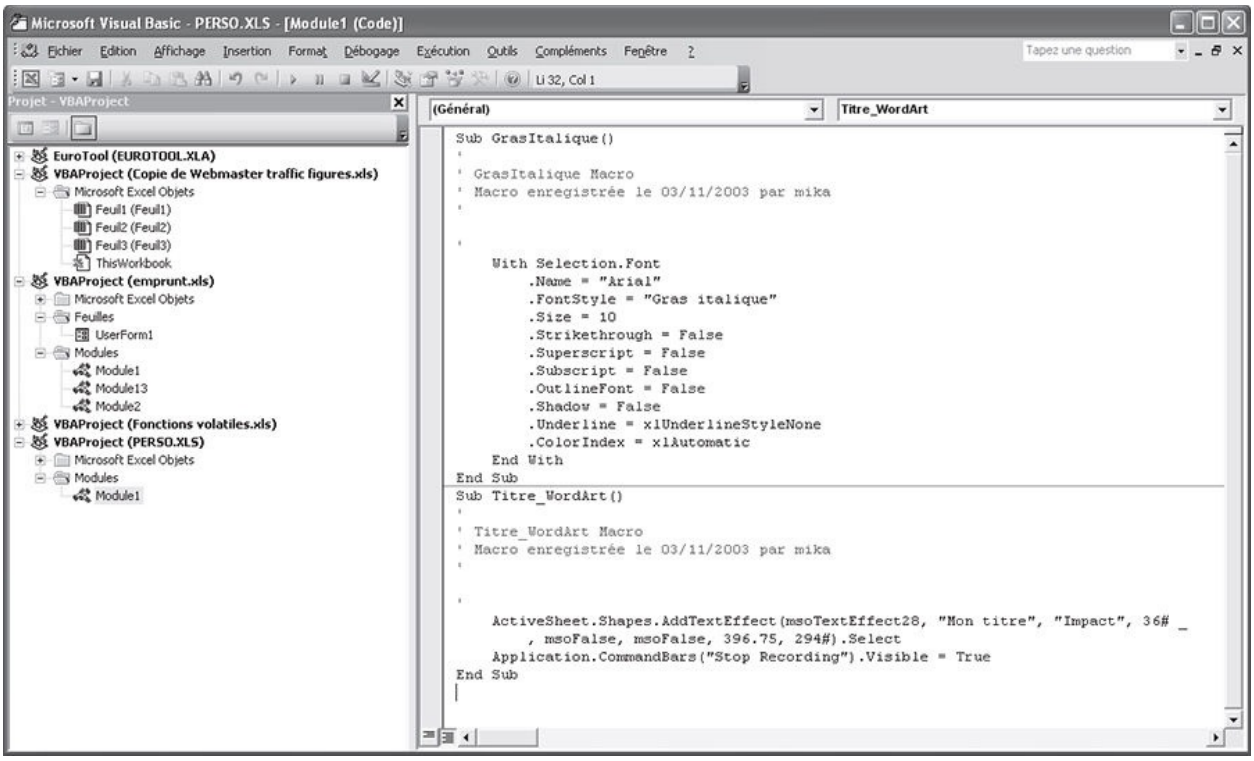


Figure 4-3 – *La fenêtre de Visual Basic Editor.*

Conseil

Dans bien des cas, l'enregistrement de macros reste la méthode la plus rapide et la plus sûre pour démarrer vos projets VBA. En laissant à l'Enregistreur de macro le soin de convertir les actions exécutées en code Visual Basic, vous êtes assuré de ne pas commettre d'erreur de saisie.

Il se peut cependant que la première étape du développement de votre projet consiste à créer une feuille permettant une interaction avec l'utilisateur. Vous accéderez alors directement à Visual Basic Editor, sans passer par la boîte de dialogue Macro.

Pour quitter Visual Basic Editor et retourner à l'application hôte, vous pouvez :

- ouvrir le menu Fichier, sélectionner la commande Fermer et retourner à Microsoft Excel ;
- taper le raccourci clavier Alt+Q ;
- cliquer sur la case de fermeture de Visual Basic Editor (située à l'extrémité supérieure droite de la fenêtre).

Pour retourner à l'application hôte sans quitter Visual Basic Editor, vous pouvez :

- cliquer sur l'icône Affichage Microsoft Excel, située à l'extrême gauche de

la barre d'outils Standard ;

- taper le raccourci clavier Alt+F11.

Les outils et les fenêtres de Visual Basic Editor

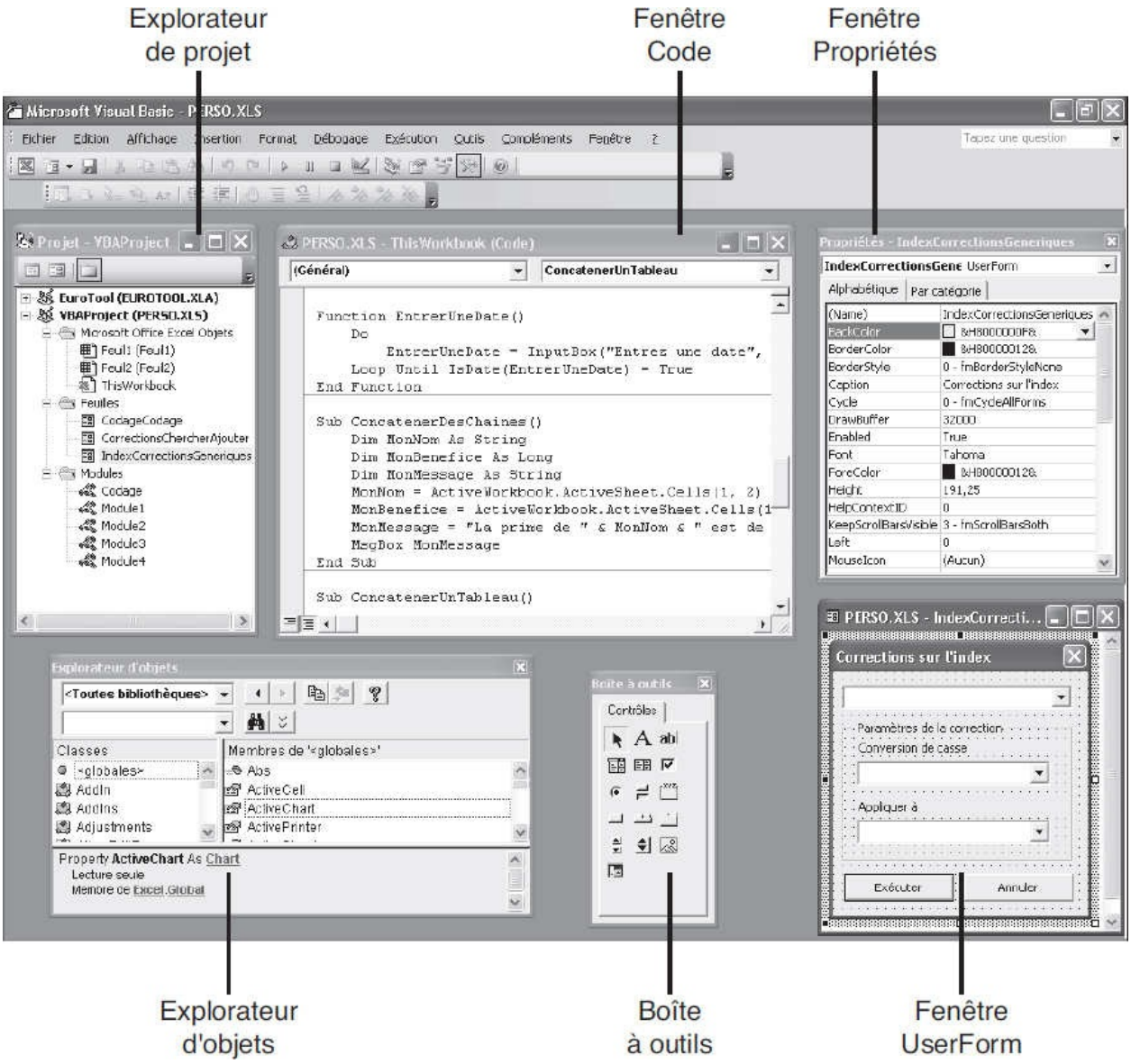


Figure 4-4 – *Visual Basic Editor.*

Cette section présente sommairement les éléments essentiels de l'interface. Vous en découvrirez plus précisément les fonctionnalités au fur et à mesure que vous avancerez dans la lecture de l'ouvrage :

- L'Explorateur de projet. Il expose les différents projets et leurs éléments

constitutifs – objets, modules, modules de classe, feuilles (ou formulaires) et Référence – et donne accès à ces éléments ou au code qui leur est attaché. Pour qu'un projet apparaisse dans l'Explorateur de projet, il faut que le document auquel il est attaché soit ouvert dans l'application hôte.

- La fenêtre Propriétés. Elle sert à visualiser et modifier l'ensemble des propriétés associées aux objets constitutifs d'un projet.
- La fenêtre Code. Vous pouvez y éditer le code de vos projets. Visual Basic Editor propose des aides à l'écriture de code et des outils de débogage.
- La fenêtre UserForm et la boîte à outils. La fenêtre UserForm est l'espace dans lequel vous concevez les feuilles VBA. La boîte à outils propose des contrôles communs tels que des cases à cocher ou des listes déroulantes à placer sur une feuille qui constituera une interface pour votre application.
- L'Explorateur d'objets. Il référence les classes, propriétés, méthodes, événements et constantes disponibles dans les bibliothèques d'objets et les procédures de votre projet. Il permet de rechercher et d'utiliser des objets que vous créez, ainsi que des objets provenant d'autres applications.

L'Explorateur de projet

L'Explorateur de projet expose les différents projets chargés dans l'application hôte et les éléments qui les composent. Il donne accès à n'importe quel élément constitutif d'un projet et autorise la création de nouveaux éléments ou, au contraire, leur suppression.

Afficher et masquer l'Explorateur de projet

Pour afficher l'Explorateur de projet, vous pouvez :

- choisir la commande Explorateur de projet du menu Affichage ;
- taper le raccourci clavier Ctrl+R ;

- cliquer sur le bouton Explorateur de projet de la barre d'outils Standard de Visual Basic Editor.

Figure 4-5 – *L'Explorateur de projet facilite l'accès aux différents éléments constitutifs d'un projet.*

Pour masquer l'Explorateur de projet, vous pouvez :

- cliquer-droit dans sa fenêtre et sélectionner la commande Masquer du menu contextuel qui s'affiche ;

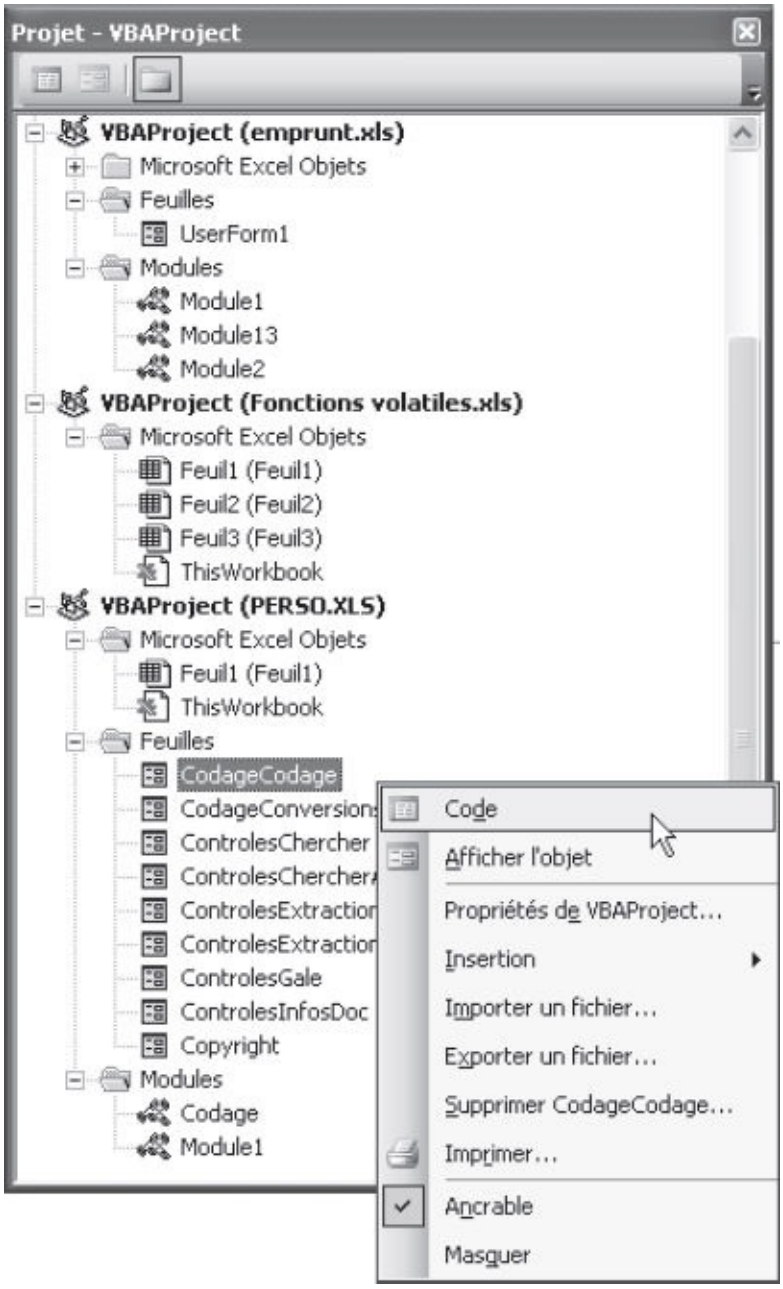


Figure 4-6 – *Le menu contextuel de l'Explorateur de projet.*

- cliquer-droit sur sa barre de titre et sélectionner la commande Fermeture ;
- taper le raccourci clavier Alt+F4 ;
- cliquer sur la case de fermeture de la fenêtre.

Naviguer dans l'Explorateur de projet

L'Explorateur de projet présente les différents projets et leurs éléments

constitutifs de façon hiérarchique. Au premier niveau apparaissent les différents projets. Sous chacun se trouvent des dossiers contenant ses éléments spécifiques :

- Microsoft Excel Objets contient les documents associés au projet. Il s'agit du classeur dans lequel est stocké le projet et de ses feuilles de calcul. Vous verrez au chapitres 15 qu'il est possible d'associer des procédures spécifiques à ces objets de façon à contrôler les interventions d'un utilisateur sur un classeur.
- Feuilles contient les feuilles (ou formulaires) du projet. Vous apprendrez à en créer aux chapitres 12 à 14.
- Modules contient les modules standards (ou modules de code) – tels que les macros – constitutifs d'un projet.
- Modules de classe contient les éventuels modules de classe d'un projet.
- Références contient les références à d'autres projets.

Attention

Si un projet ne contient aucun module ou module de classe, les dossiers correspondants n'apparaîtront pas dans l'Explorateur de projet.

Info

Lorsque vous créez une macro, elle est enregistrée dans un nouveau module accessible *via* le dossier Modules du projet correspondant. Son nom est Module1, Module2 si Module1 existe déjà, etc.

Le signe plus (+) développe l'arborescence d'un dossier ou d'un projet et le signe moins (−) la réduit.

Plutôt que par dossiers, les éléments constitutifs d'un projet peuvent être présentés par ordre alphabétique. Cliquez simplement sur le bouton Basculer dossiers de l'Explorateur de projet. La figure 4-7 présente l'aspect après masquage des dossiers. Pour revenir à l'affichage précédent, cliquez de nouveau sur ce bouton.

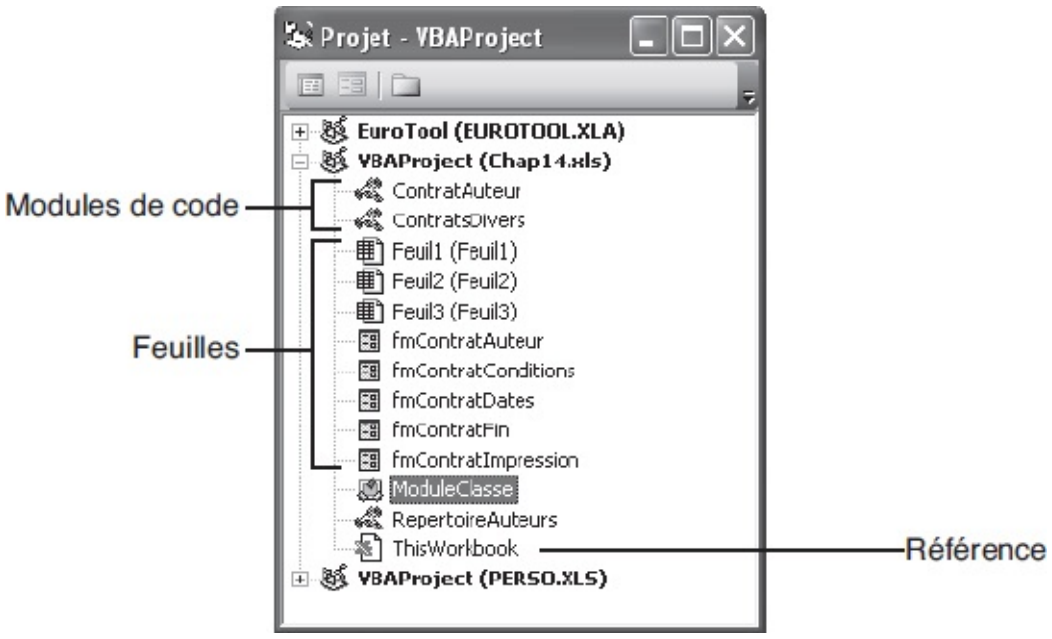


Figure 4-7 – *Lorsque l'affichage des dossiers est désactivé, les icônes distinguent les éléments du projet.*

Accéder aux objets et au code des projets

Outre le bouton Basculer dossiers, l'Explorateur de projet présente deux boutons facilitant l'accès au code et aux objets constitutifs d'un projet :

	Afficher le code	Affiche le code de l'élément sélectionné dans l'Explorateur de projet pour l'écrire ou le modifier.
	Afficher l'objet	Affiche l'objet sélectionné dans l'Explorateur de projet. Il peut s'agir d'une feuille (dossier UserForm) ou d'un document. Ce bouton est désactivé si l'objet sélectionné est un module de code.

L'Explorateur d'objets

Lorsqu'on commence à développer en VBA, la difficulté essentielle consiste à manipuler les objets de l'application hôte (dans notre cas les objets Excel). Comment, par exemple, accéder à une plage de cellules d'une feuille spécifique d'un classeur et y insérer une formule ? L'Enregistreur de macro est dans de nombreux cas la solution à ce problème. Vous manipulez les objets Excel après avoir activé l'Enregistreur de macro, puis vous visualisez dans Visual Basic Editor les mots-clés Visual Basic utilisés pour accéder aux objets, à leurs propriétés et à leurs méthodes.

Cependant, certains éléments de code d'un programme VBA ne peuvent être

générés à l'aide de l'Enregistreur de macro et doivent être saisis dans la fenêtre Code du programme. Il vous faut alors connaître la position de l'objet auquel vous souhaitez accéder dans la hiérarchie de classes de l'application. Vous devez aussi connaître les méthodes et propriétés associées à cet objet pour pouvoir le manipuler ou en extraire des informations.

Les chapitres précédents vous ont initié à la syntaxe VBA permettant d'accéder à un objet. Pour autant, lorsque vous commencerez à développer dans Visual Basic Editor, vous ne connaîtrez pas toujours le chemin à emprunter pour accéder à tel ou tel objet, ni la méthode à lui appliquer pour effectuer telle ou telle opération. L'Explorateur d'objets constitue pour cela une aide très appréciable pour le développeur, en supplément de l'Aide de VBA. Il recense en effet l'ensemble des objets disponibles dans les *bibliothèques d'objets* accessibles pour un projet, ainsi que les propriétés, constantes, méthodes et événements associés.

Définition

Une bibliothèque d'objets est un fichier contenant toutes les données des objets (leurs propriétés, méthodes, événements, constantes, etc.). Ce fichier porte l'extension .OLB, et c'est à lui que se réfère Visual Basic lorsque vous manipulez des objets Excel. Le nom de fichier de la bibliothèque d'objets d'Excel ainsi que son emplacement varient d'une version à l'autre. Pour le localiser, effectuez une recherche sur ***.olb***.

Lorsque vous recherchez un objet ou souhaitez en connaître les membres – c'est ainsi que l'on nomme les éléments Visual Basic (méthodes, propriétés, événements, constantes) associés –, l'Explorateur d'objets vous fournit une documentation complète. Il donne accès au modèle d'objets de l'application hôte, mais aussi à ceux d'autres applications et aux objets, procédures et constantes que vous avez créés dans le cadre de votre projet, ainsi qu'aux rubriques d'aide associées à chacun de ces éléments.

Afficher et masquer l'Explorateur d'objets

Pour afficher l'Explorateur d'objets, vous pouvez :

- sélectionner la commande Explorateur d'objets du menu Affichage ;
- taper le raccourci clavier F2 ;

- cliquer sur le bouton Explorateur d'objets de la barre d'outils Standard de Visual Basic Editor.

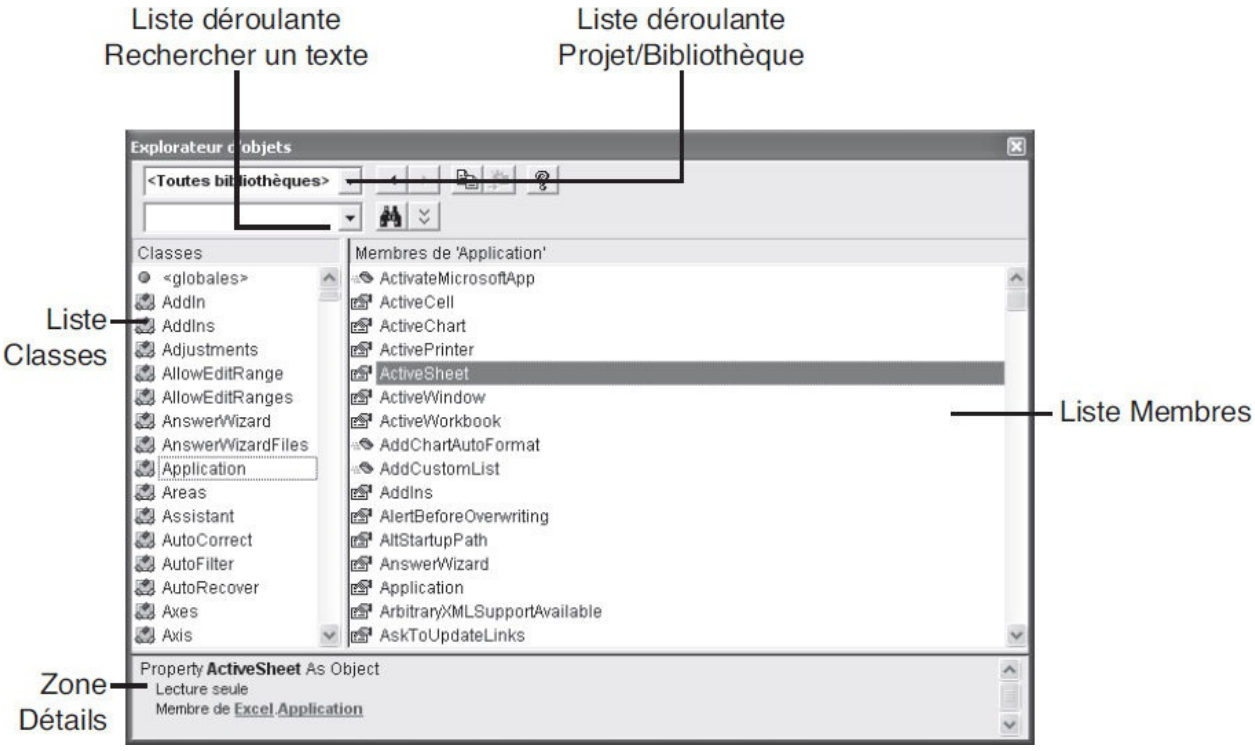


Figure 4-8 – *L'Explorateur d'objets sert à explorer l'ensemble des objets disponibles pour un projet.*

Pour masquer l'Explorateur d'objets, vous pouvez :

- cliquer-droit dans sa fenêtre et sélectionner la commande Masquer du menu contextuel qui s'affiche ;
- cliquer sur l'icône située à gauche de la barre de titre et sélectionner la commande Fermeture ;
- cliquer sur la case de fermeture de la fenêtre.

Naviguer dans l'Explorateur d'objets

S'il peut effrayer le programmeur novice, l'Explorateur d'objets est en réalité d'une utilisation simple et intuitive. Cette section en présente l'utilisation.

La liste Projet/Bibliothèque

La liste déroulante Projet/Bibliothèque sert à sélectionner le projet ou la bibliothèque d'objets de votre choix. Le tableau 4-1 présente les bibliothèques les plus courantes.

Tableau 4-1. Les bibliothèques les plus courantes de l'Explorateur d'objets

Bibliothèque	Description
<Toutes bibliothèques>	Lorsque cette option est sélectionnée, les objets sont affichés, toutes bibliothèques confondues.
MSForms	Contient les objets accessibles dans la fenêtre UserForm, tels que les boutons d'options, cases à cocher, listes déroulantes, etc.
Office	Contient les objets Microsoft Office. Il s'agit des objets communs aux applications Office.
VBA	Il s'agit de la bibliothèque Visual Basic pour Applications. Les objets y sont classés par thème. Par exemple, le module `Information` contient les procédures pour renvoyer et vérifier des informations et le module `String` contient les procédures effectuant des opérations sur des chaînes de caractères.
Excel	Contient les objets d'Excel.
Autres applications	Contient les objets des autres applications référencées dans votre projet. Référencer une autre application donne accès à ses objets à partir d'Excel. Vous verrez au chapitres 6 comment créer une référence à la bibliothèque d'objets d'une autre application et comment en manipuler les objets.
Projets	Affiche les objets propres au projet, tels que les feuilles, les modules de classe et les modules de code que vous avez créés.

Les zones Classes et Membres de

Lorsque vous sélectionnez un item dans la liste déroulante Bibliothèque/Projet, la zone Classes affiche l'ensemble des classes disponibles dans cette bibliothèque. Elles sont affichées par type et par ordre alphabétique au sein de chaque type. Chaque type de classe (feuille, module de code, module de classe) est symbolisé par une icône. Lorsqu'une classe contient du code rédigé par l'utilisateur, son nom apparaît en gras (voir figure 4-9).

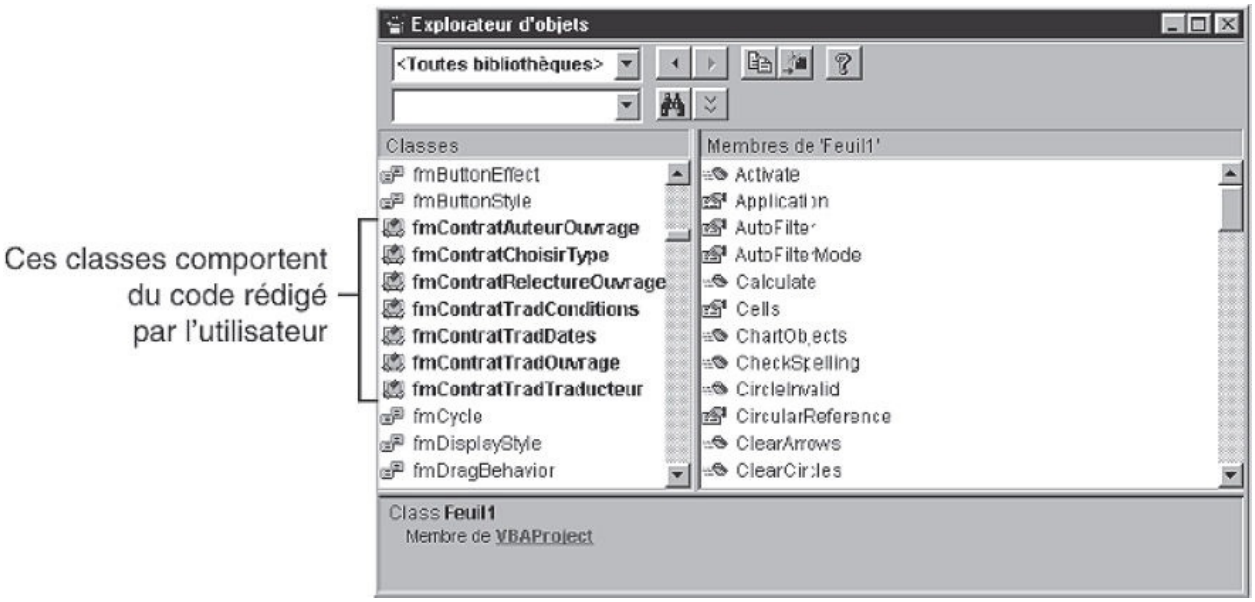


Figure 4-9 – *Les classes contenant du code apparaissent en gras.*

Lorsque vous sélectionnez une classe dans la zone de gauche, tous ses membres apparaissent dans celle de droite, affichés par type (propriétés, constantes, méthodes et événements) et par ordre alphabétique au sein de chaque type. Chaque type de membre est symbolisé par une icône. Lorsqu'un membre contient du code rédigé par l'utilisateur, son nom apparaît en gras.

Définition

On appelle membres d'une classe l'ensemble des éléments référencés pour cette classe, c'est-à-dire ses propriétés, constantes, méthodes et événements.

Astuce

Pour afficher les membres d'une classe par ordre alphabétique, indépendamment de leur type, cliquez-droit dans l'Explorateur d'objets et, dans le menu contextuel qui s'affiche, sélectionnez la commande Membres du groupe. Pour revenir à un affichage par groupe, répétez cette opération.

Accéder à la rubrique d'aide de l'objet sélectionné

Lorsqu'un élément est sélectionné dans l'Explorateur d'objets, il est très simple d'accéder à la rubrique d'aide correspondante par l'une des méthodes suivantes :

- cliquer sur le bouton Aide de l'Explorateur d'objets ;
- taper le raccourci clavier F1 ;
- cliquer-droit sur l'élément voulu et sélectionner la commande Aide du menu contextuel.

Vous obtenez ainsi une aide précieuse sur l'utilisation de l'élément voulu (voir figure 4-10). Vous pouvez copier l'exemple fourni afin de le coller dans votre propre code.

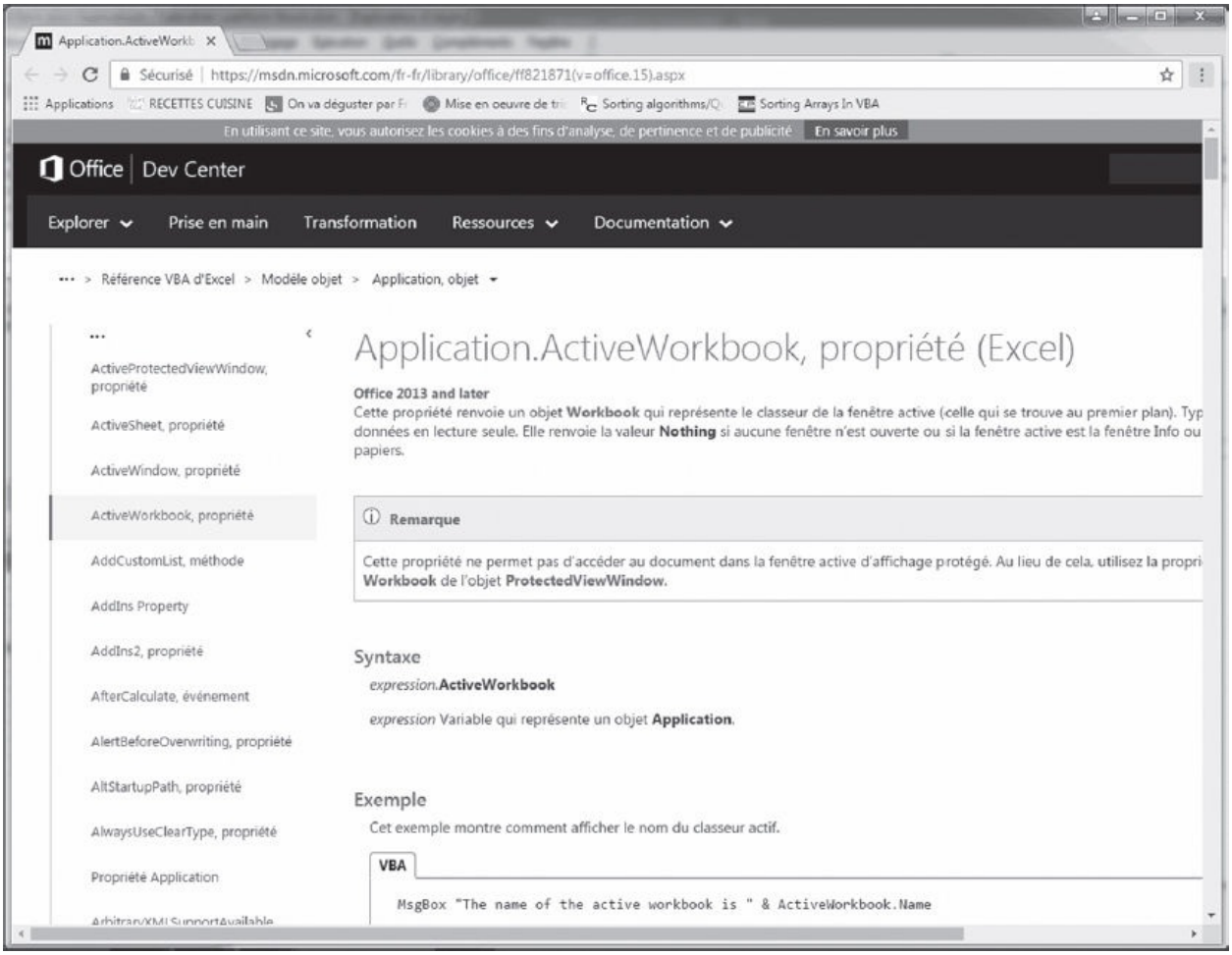

Application.ActiveWorkb
Sécurisé | https://msdn.microsoft.com/fr-fr/library/office/ff821871(v=office.15).aspx
Applications
RECETTES CUISINE
On va déguster par Fr
Mise en oeuvre de tri
Sorting algorithms/Q
Sorting Arrays In VBA
En utilisant ce site, vous autorisez les cookies à des fins d'analyse, de pertinence et de publicité
En savoir plus
Office | Dev Center
Explorer
Prise en main
Transformation
Ressources
Documentation
Référence VBA d'Excel > Modèle objet > Application, objet
ActiveProtectedViewWindow, propriété
ActiveSheet, propriété
ActiveWindow, propriété
ActiveWorkbook, propriété
AddCustomList, méthode
AddIns Property
AddIns2, propriété
AfterCalculate, événement
AlertBeforeOverwriting, propriété
AltStartupPath, propriété
AlwaysUseClearType, propriété
Propriété Application
Application.ActiveWorkbook, propriété (Excel)
Office 2013 and later
Cette propriété renvoie un objet Workbook qui représente le classeur de la fenêtre active (celle qui se trouve au premier plan). Typ
données en lecture seule. Elle renvoie la valeur Nothing si aucune fenêtre n'est ouverte ou si la fenêtre active est la fenêtre Info ou
papiers.
Remarque
Cette propriété ne permet pas d'accéder au document dans la fenêtre active d'affichage protégé. Au lieu de cela, utilisez la propri
Workbook de l'objet ProtectedViewWindow.
Syntaxe
expression.ActiveWorkbook
expression Variable qui représente un objet Application.
Exemple
Cet exemple montre comment afficher le nom du classeur actif.
VBA
MsgBox "The name of the active workbook is " & ActiveWorkbook.Name

Figure 4-10 – *L'Explorateur d'objets facilite l'accès aux rubriques d'aide des objets affichés.*

Les autres contrôles de l'Explorateur d'objets

La zone Détails affiche un bref descriptif du membre sélectionné (voir figure 4-11). Si aucun membre n'est sélectionné, la description concerne la classe. Enfin, si aucune classe n'est sélectionnée, cette zone indique le chemin d'accès à la bibliothèque ou au projet affiché dans l'Explorateur d'objets.

Cette zone présente aussi, sous la forme d'un hyperlien de couleur verte, la position de l'objet sélectionné dans le modèle d'objets et les éléments du langage éventuellement liés. Cliquer sur un de ces hyperliens affiche les membres de l'objet correspondant.

Le bouton Retourner ramène aux sélections précédentes dans les listes Classes et Membres de. Chaque clic vous fait remonter d'une sélection. Le bouton

Avancer revient aux dernières sélections effectuées après avoir utilisé le bouton Retourner.

Le bouton Copier place dans le Presse-papiers le texte sélectionné. Il peut s'agir du texte de la zone Détails, de la zone Membres de ou de la zone Classes. Le texte ainsi copié peut être collé dans une fenêtre Code.

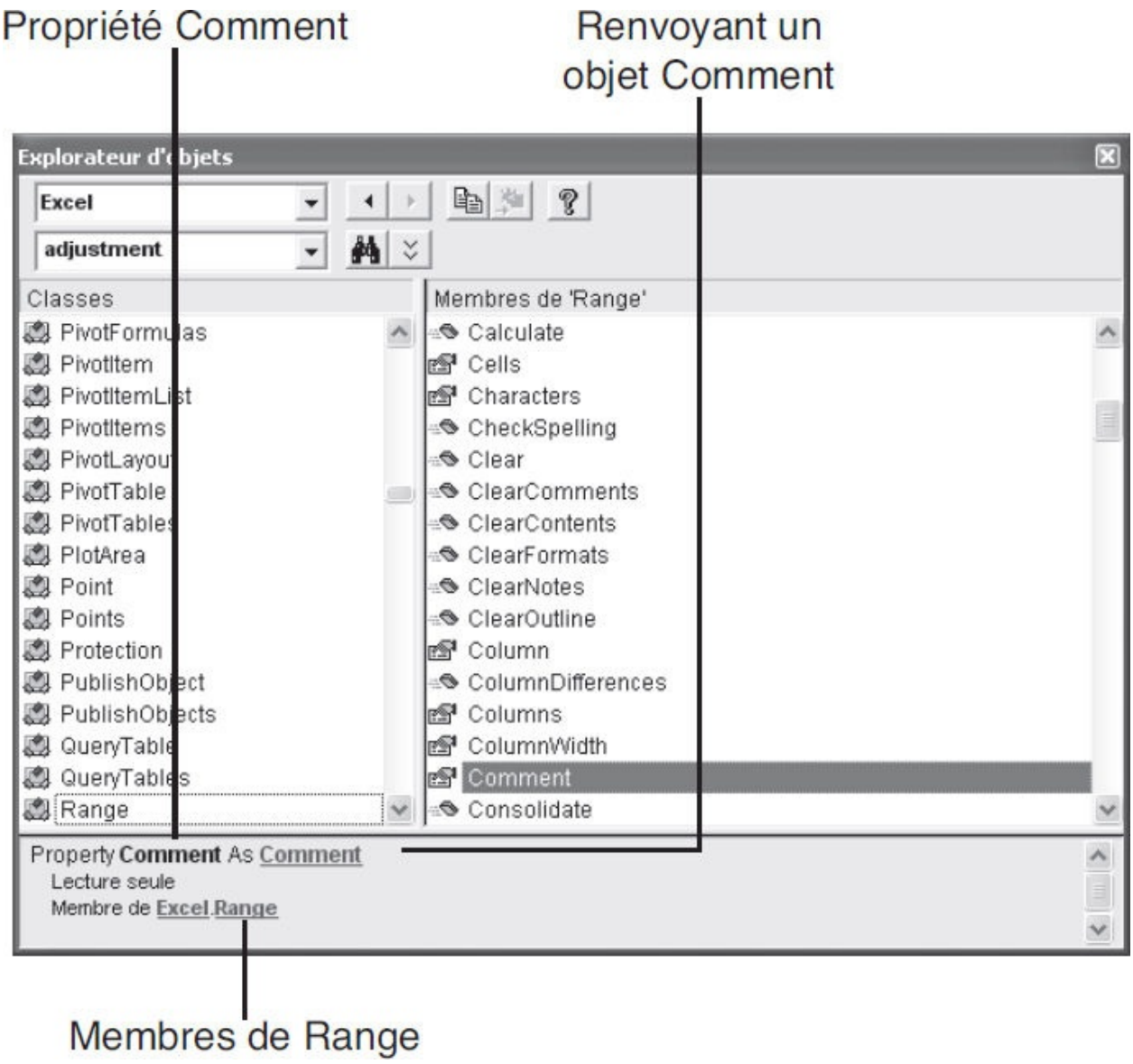


Figure 4-11 – *La zone Détails récapitule les données propres à l'élément sélectionné.*

Le bouton Afficher la définition est accessible lorsque la classe ou le membre de classe sélectionné contient du code rédigé. Il entraîne l'affichage de la fenêtre Code correspondant à la sélection en cours dans l'Explorateur d'objets.

Rechercher du texte

L'Explorateur d'objets offre un outil pour rechercher des chaînes de caractères dans les bibliothèques de votre choix. Procédez comme suit :

1. Dans la zone Bibliothèque/Projet, sélectionnez la bibliothèque ou le projet dans lequel vous souhaitez que la recherche s'effectue.

   Si vous ne savez pas où chercher, sélectionnez <Toutes bibliothèques>.

2. Dans la zone Rechercher texte, saisissez la chaîne de caractères que vous voulez.

Astuce

Si vous n'êtes pas certain de l'orthographe du texte à rechercher, utilisez les caractères génériques suivants :

- * toute chaîne ;
- ? tout caractère.

3. Cliquez sur le bouton Rechercher. La zone Résultats de la recherche s'affiche. Pour chacun des éléments trouvés, la bibliothèque, la classe et le membre sont indiqués (voir figure 4-12).
4. Si vous souhaitez afficher les résultats pour d'autres bibliothèques, modifiez simplement la sélection dans la zone Bibliothèque/Projet. Les résultats sont automatiquement mis à jour.

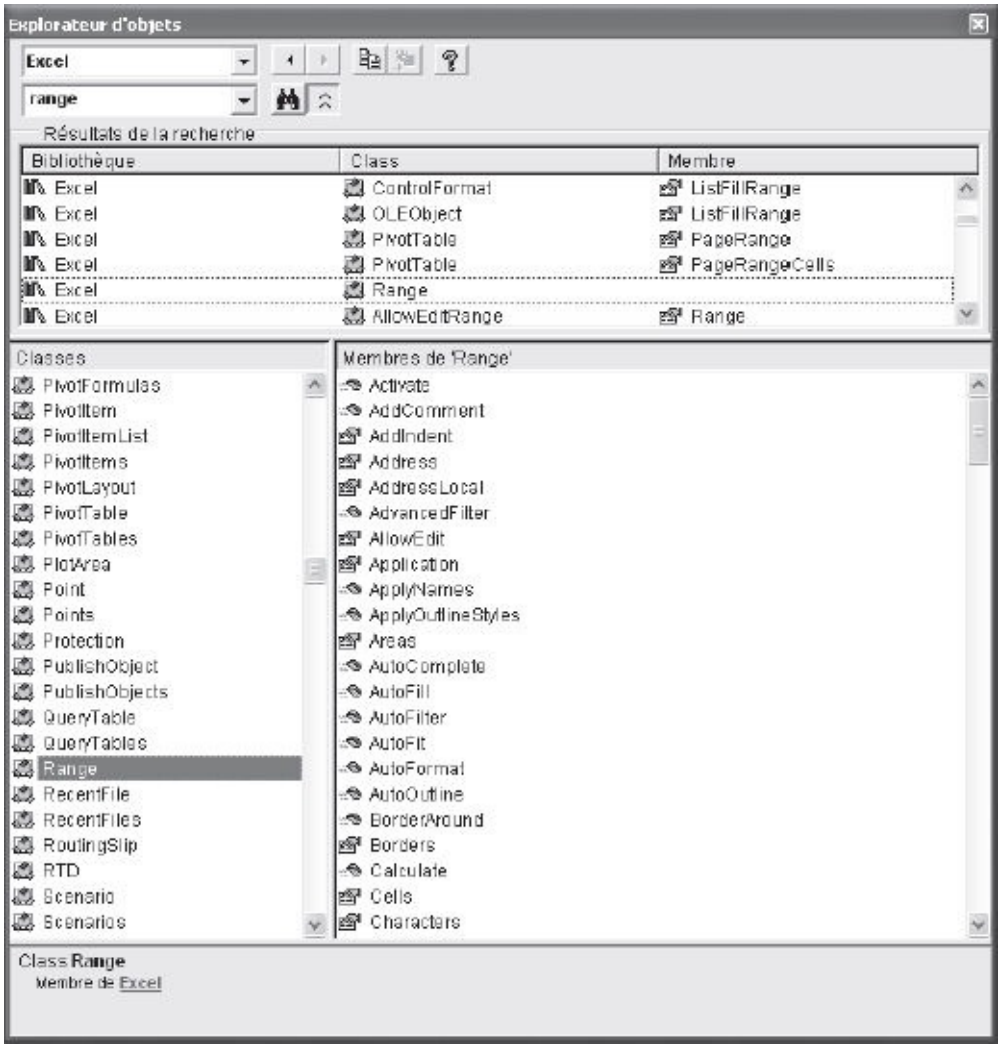


Figure 4-12 – *La zone Résultat de la recherche liste l'ensemble des éléments trouvés contenant la chaîne spécifiée.*

5. Pour afficher les informations concernant l'un des résultats dans les zones Classe et Membres de, sélectionnez l'élément voulu dans la zone Résultat de la recherche.

   Les zones Classe et Membres de sont automatiquement mises à jour, les objets concernés par la sélection étant entourés de pointillés (voir figure 4-13).

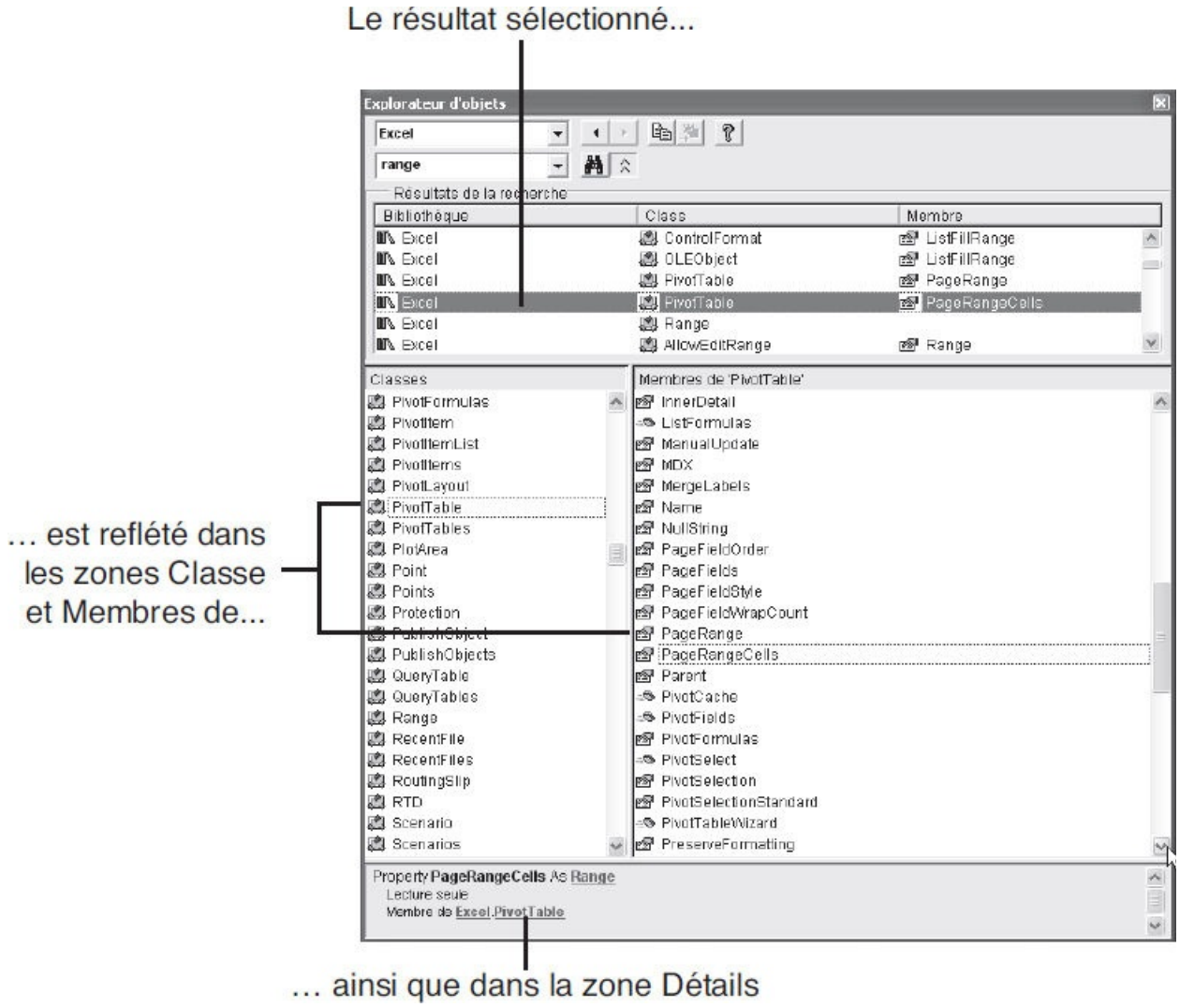


Figure 4-13 – *Sélectionnez l'élément qui vous intéresse dans la zone Résultat de la recherche.*

6. Pour ouvrir la rubrique d'aide associée à l'un des éléments trouvés, sélectionnez-le et cliquez sur le bouton Aide, ou tapez le raccourci clavier F1.

7. Votre recherche terminée, vous pouvez choisir de masquer la zone de résultats. Pour cela, cliquez simplement sur le bouton Afficher/Masquer les résultats de la recherche.

   Pour afficher de nouveau cette zone sans lancer une nouvelle recherche, cliquez une nouvelle fois sur le bouton.

La fenêtre UserForm

La fenêtre UserForm sert à dessiner des boîtes de dialogue pour vos projets. Dans Visual Basic pour Applications, ces boîtes sont appelées *feuilles* – on parle aussi de *formulaires*. Une feuille peut être très simple, ou présenter un grand nombre de fonctionnalités (figure 4-14).

Le développement de feuilles est un aspect essentiel de la programmation VBA. Les chapitres 12 à 14 sont entièrement consacrés à ce sujet. Cette section présente sommairement les possibilités de développement dans la fenêtre UserForm.

On peut distinguer deux phases essentielles dans la création d'une interface utilisateur :

- le développement visuel de la feuille – traité au chapitres 12 ;
- l'association de code aux différents éléments de la feuille – traitée aux chapitres 13 et 14.

Dates de remise et de parution
Indiquez la date à valider et sélectionnez-la sur le calendrier.
Date de remise :
Date de parution :
Date à valider :
Date de remise
December 2003
December
2003
lun. mar. mer. jeu. ven. sam. dim.
24 25 26 27 28 29 30
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31 1 2 3 4
Annuler
Retour
Terminer

Figure 4-14 – *Une feuille développée dans Visual Basic Editor.*

La première phase consiste à *dessiner* la feuille, c'est-à-dire à créer l'interface dont vous avez besoin pour votre application. Cette étape consiste essentiellement à placer des *contrôles* sur la feuille et à en déterminer les positions respectives. Il s'agit des éléments constitutifs d'une fenêtre tels qu'une case à cocher, une liste déroulante ou un bouton de commande permettant une intervention de l'utilisateur. Un contrôle est un objet ; en tant que tel, il possède des propriétés, des méthodes et des événements définis.

Par exemple, un contrôle `Checkbox` (case à cocher) possède une propriété `Value` indiquant son état (`True` si la case est cochée, `False` si elle est décochée et `Null` si elle n'est ni cochée ni décochée – grisée). La méthode `SetFocus` lui affecte le focus, c'est-à-dire en fait l'objet recevant les événements clavier ou souris. Enfin, un événement `Click` (un clic de souris) affectant cet objet peut être détecté et entraîner un comportement spécifique de l'application, tel que le déclenchement d'une procédure (appelée procédure d'événement).

Certains contrôles, dits interactifs, réagissent aux actions de l'utilisateur (un bouton OK déclenchant la validation des données de la feuille et le lancement d'une procédure) ; les autres, dits statiques, ne sont accessibles qu'au niveau du code (un intitulé sur la feuille ne pouvant être modifié par l'utilisateur).

La figure 4-15 présente la boîte de dialogue Police de Word, sur laquelle vous pouvez visualiser différents types de contrôles.

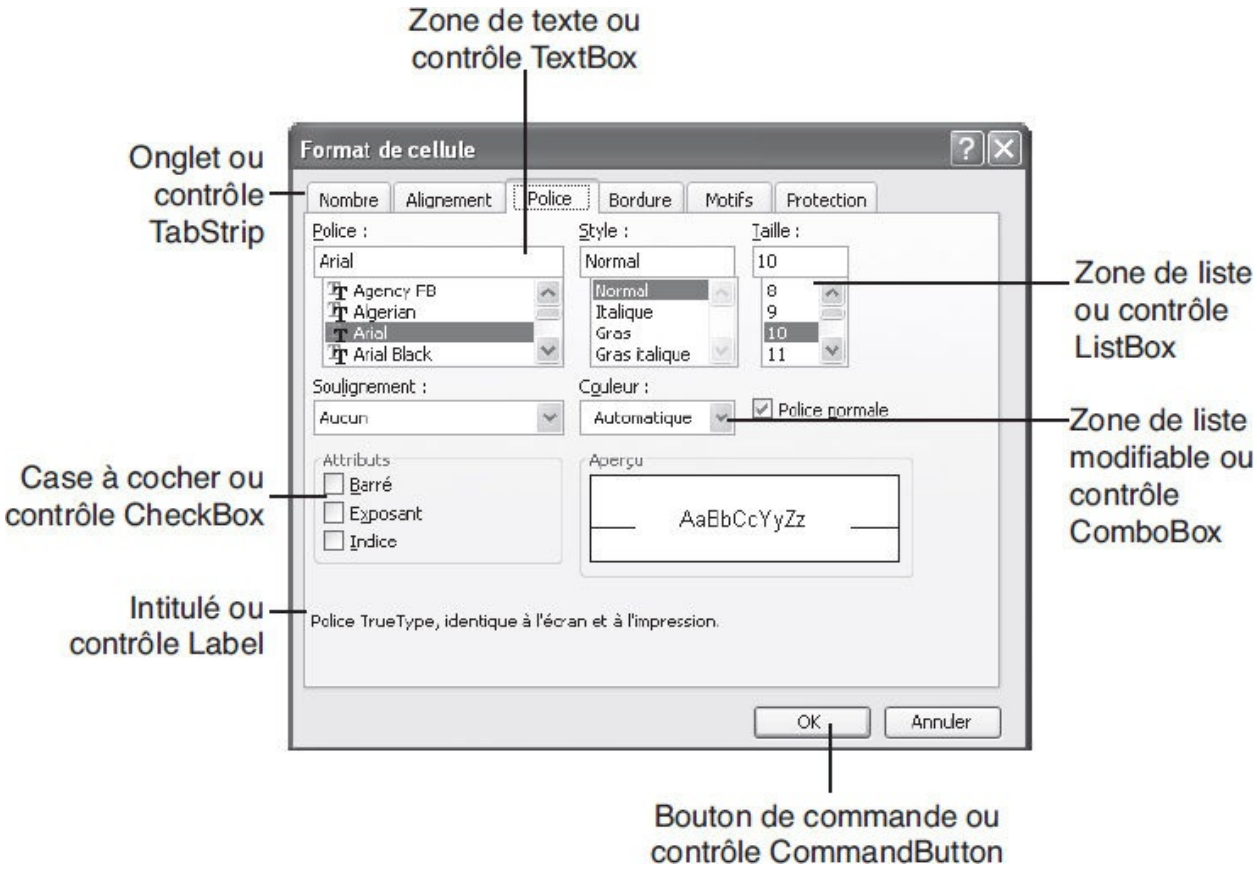


Figure 4-15 – *Les contrôles à placer sur une feuille sont les mêmes que ceux que vous rencontrez dans les applications Office.*

Par défaut, la boîte à outils propose les contrôles les plus couramment rencontrés dans l'application hôte (voir figure 4-16). Vous verrez au chapitres 12 comment la personnaliser en y ajoutant des contrôles ou en en modifiant l'ordonnancement.

Figure 4-16 – *Les contrôles de la boîte à outils sont faciles à déposer sur une feuille.*

Afficher et masquer la fenêtre UserForm et la boîte à outils

Pour afficher la fenêtre UserForm, vous devez soit ouvrir une feuille dans un projet (accessible dans le dossier Feuilles de l'Explorateur de projet), soit créer une nouvelle feuille. Pour quitter la fenêtre UserForm, cliquez sur la case de fermeture située dans l'angle supérieur droit de la fenêtre.

Pour afficher/masquer la boîte à outils, une fenêtre UserForm doit être active :

- sélectionnez la commande Boîte à outils du menu Affichage ;

ou

- cliquez sur le bouton Boîte à outils de la barre d'outils Standard de Visual Basic Editor.

Vous verrez au chapitres 12 que Visual Basic Editor propose des outils facilitant grandement la phase de développement visuel des feuilles.

La fenêtre Code

La fenêtre Code est l'éditeur de Visual Basic Editor. Elle sert à écrire, visualiser et modifier des programmes Visual Basic. Toute action qu'exécute

une application VBA existe sous forme de code et l'ensemble des instructions qui constituent un projet est édité dans cette fenêtre.

Une fenêtre Code est toujours attachée à l'un des modules apparaissant dans l'Explorateur de projet (voir figure 4-17). Il peut s'agir d'un module standard, d'un module de classe, d'une feuille ou du document auquel est attaché le projet.

La fenêtre Code présente de nombreuses et précieuses fonctionnalités, destinées à faciliter le développement de vos projets. Vous pouvez, par exemple, en ouvrir une pour chacun des modules apparaissant dans votre projet et copier, coller ou déplacer des instructions d'une fenêtre et d'une procédure à l'autre, rechercher des chaînes de caractères ou marquer certains endroits du programme en y plaçant des signets, paramétrer la fenêtre afin que la syntaxe Visual Basic soit vérifiée au fur et à mesure de la saisie du code, etc.

Afficher et masquer la fenêtre Code

Pour accéder au code d'un élément constitutif d'un projet, ouvrez l'Explorateur de projet et sélectionnez-y l'élément qui vous intéresse. Effectuez ensuite l'une des opérations suivantes :

- Double-cliquez sur un module de code.
- Cliquez-droit sur l'élément et choisissez la commande Code du menu contextuel.
- Cliquez sur le bouton Afficher le code de l'Explorateur de projet.
- Sélectionnez la commande Code du menu Affichage.
- Tapez le raccourci clavier F7.

Info

Pour accéder au code des contrôles d'une feuille, double-cliquez dessus ou cliquez-droit et choisissez la commande Code du menu contextuel qui s'affiche.

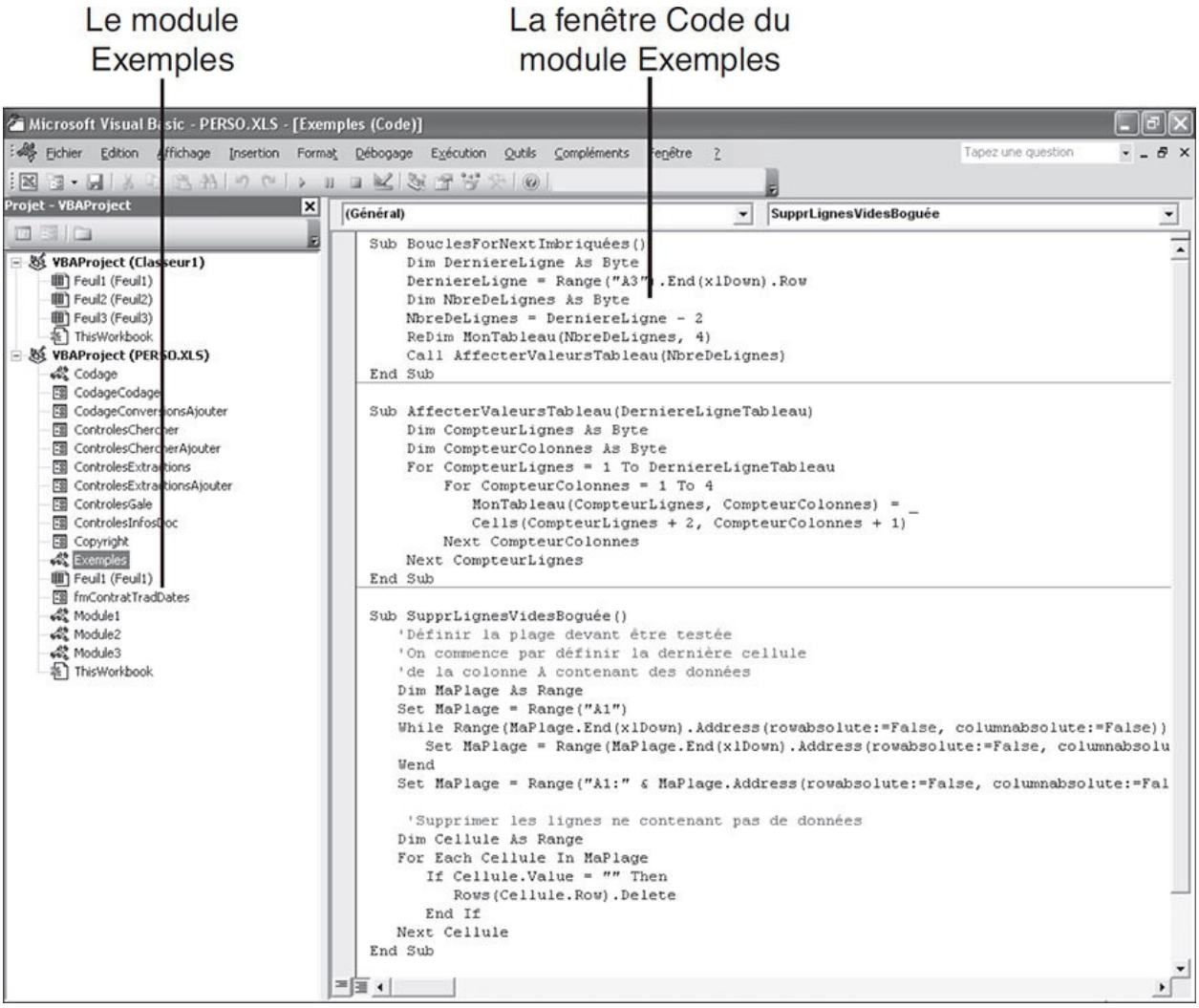


Figure 4-17 – *Chacun des modules d'un projet possède une fenêtre Code.*

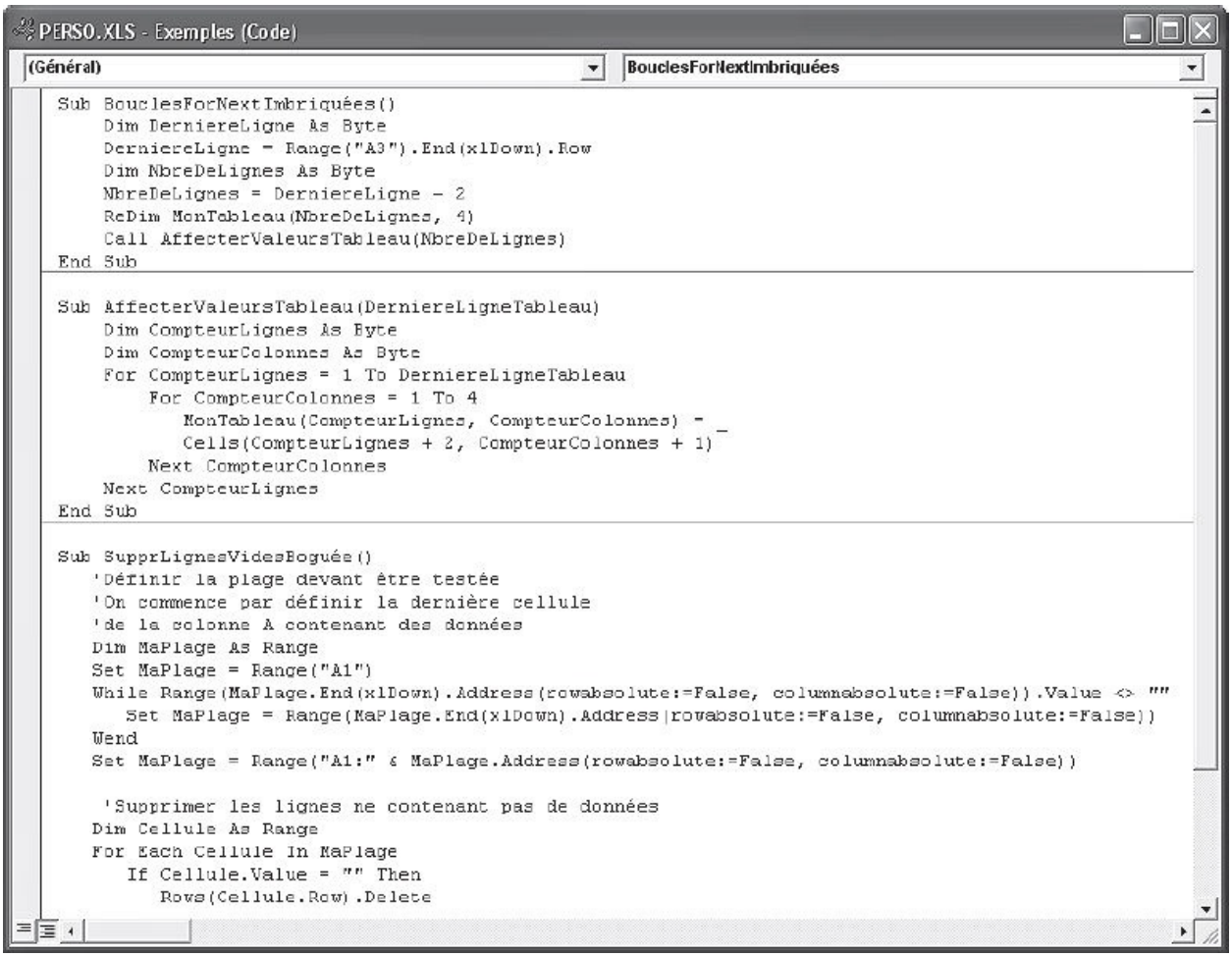


Figure 4-18 – *La fenêtre Code est le cœur de Visual Basic Editor.*

Pour masquer la fenêtre Code, cliquez sur la case de fermeture ou cliquez-droit dans la fenêtre et sélectionnez la commande Masquer du menu contextuel qui s'affiche.

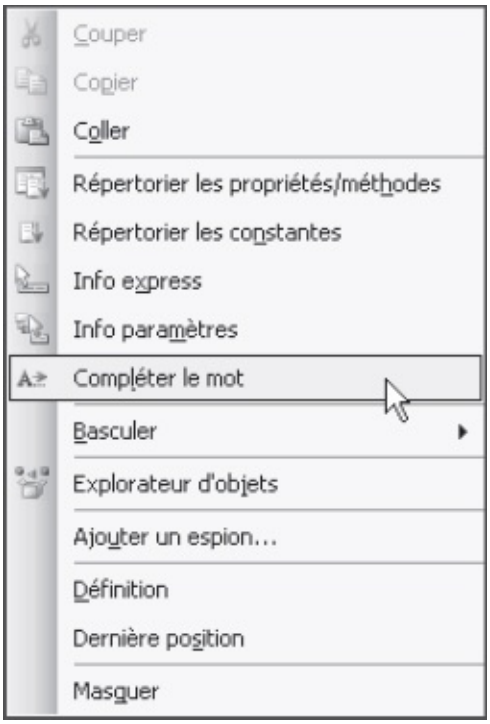


Figure 4-19 – *Le menu contextuel de la fenêtre Code.*

Naviguer dans la fenêtre Code

Par défaut, la fenêtre Code affiche toutes les *procédures* constituant le module par ordre alphabétique, séparées par des lignes grises. Un même module pouvant contenir un nombre important de procédures, deux zones de listes sont disponibles pour se déplacer rapidement dans le code.

Définition

Une procédure est un bloc d'instructions, tel qu'une macro, délimité par des mots-clés d'encadrement et exécuté en tant qu'entité. Vous verrez au chapitres 5 qu'une procédure peut avoir pour fonction de renvoyer une valeur ou d'exécuter un certain nombre d'actions. Le code exécutable des projets VBA est toujours contenu dans des procédures.

Les listes déroulantes Objet et Procédure

La liste déroulante Objet référence tous les objets contenus dans le module (dans le cas d'une feuille, tous les contrôles). Lorsque vous sélectionnez un objet dans cette liste, son code apparaît dans le haut de la fenêtre, le point d'insertion étant placé sur la première ligne (voir figure 4-20).

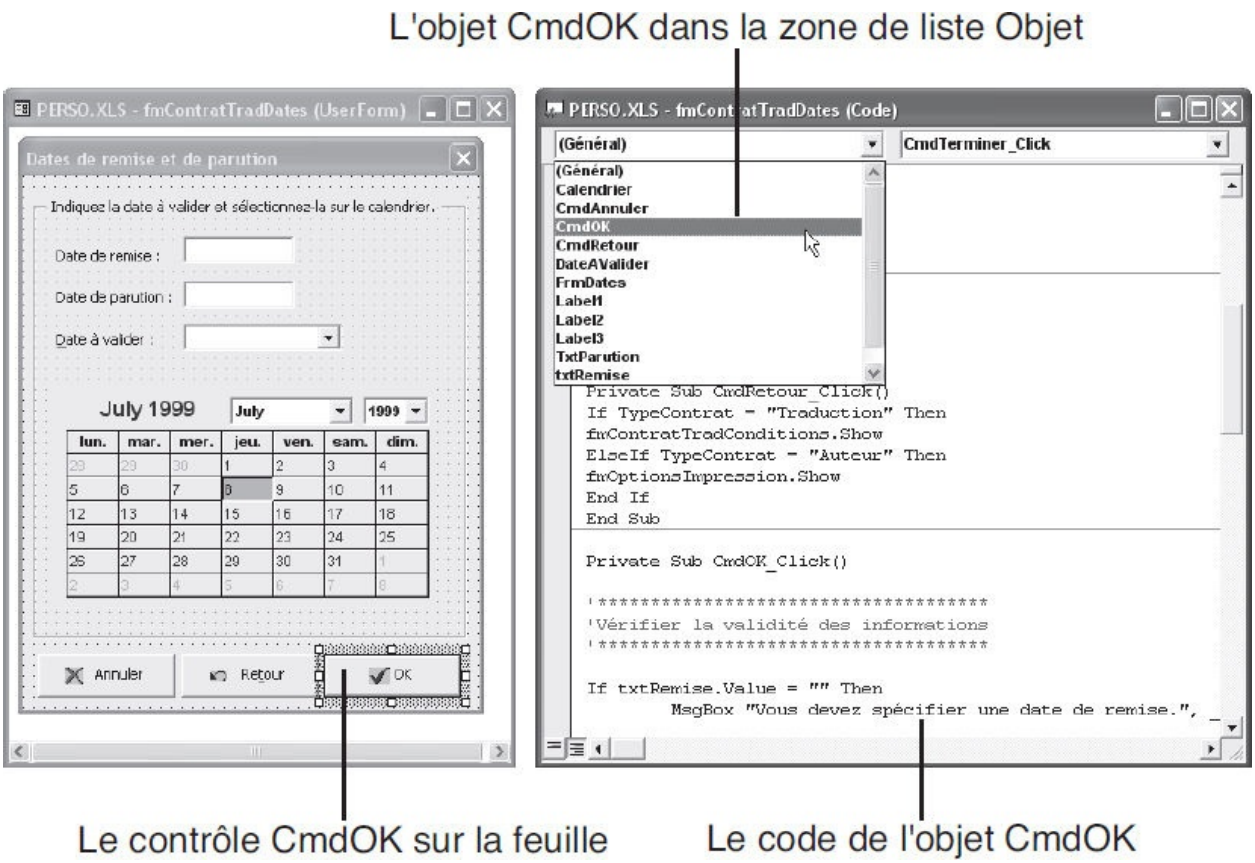


Figure 4-20 – *La liste Objet répertorie l'ensemble des objets du module.*

Les éléments référencés dans la liste déroulante Procédure peuvent être des procédures ou des événements, selon le type du module. La figure 4-21 présente la liste Procédure dans la fenêtre Code du module LivreExcel ; toutes les procédures du projet y sont référencées, séparées par des lignes grises.

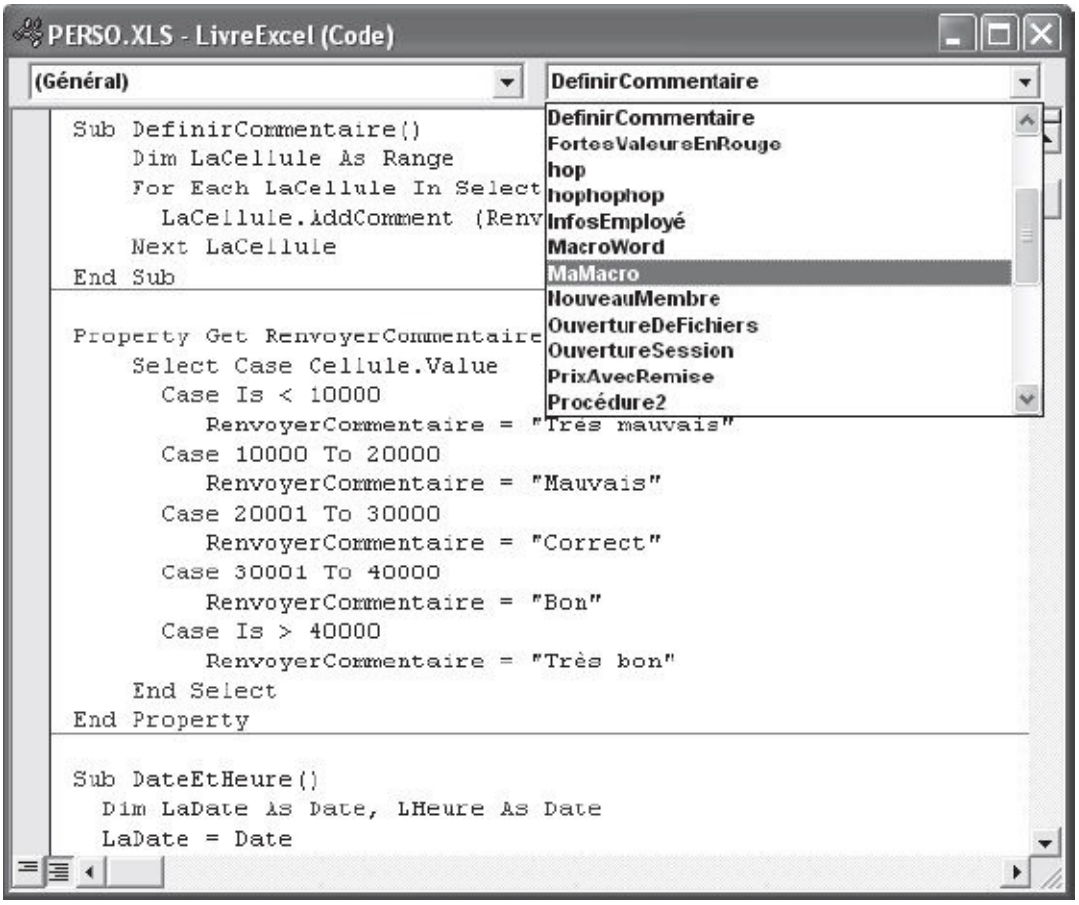


Figure 4-21 – *Utilisez cette zone pour accéder rapidement à une procédure du module.*

Rappel

Lorsque vous enregistrez ou créez des macros dans Excel, elles sont stockées dans un module nommé Module*n*, accessible dans le dossier Modules de l'Explorateur de projet.

Dans le haut de la liste se trouve l'entrée (Déclarations). Il s'agit des *déclarations générales* du module. Si la fenêtre Code est celle d'une feuille, (Général) doit être sélectionné dans la liste Objet pour que (Déclarations) apparaisse dans la liste Procédure. Cette dernière référence l'ensemble des événements (un clic de souris, une saisie au clavier, etc.) reconnus pour la feuille ou le contrôle sélectionné dans la liste Objet (voir figure 4-22) et la fenêtre Code affiche la procédure d'événement correspondante. Si l'option (Général) est sélectionnée dans la liste Objet, la liste Procédure répertorie l'ensemble des déclarations et des procédures de la feuille.

Définition

Une procédure d'événement (ou événementielle) est une procédure attachée à un événement spécifique, tel qu'un clic de souris. Elle s'exécute lorsque l'événement est reconnu par le programme. Ce sujet est traité au chapitres 14.

Lorsque vous sélectionnez une procédure dans la liste, son code apparaît dans le haut de la fenêtre Code, le point d'insertion étant automatiquement placé sur la première ligne.

Figure 4-22 – *La liste déroulante Procédure répertorie l'ensemble des événements référencés pour un objet.*

Sélection de texte dans la fenêtre Code

Lorsque vous vous trouvez dans une fenêtre Code, vous pouvez sélectionner du texte à l'aide de la souris ou du clavier, de façon classique. Il existe en outre des modes de sélection spécifiques à cette fenêtre de Visual Basic Editor.

- Pour sélectionner une procédure à l'aide de la souris, placez le pointeur sur la gauche du code de façon à ce qu'il prenne la forme d'une flèche orientée

vers la droite et double-cliquez (voir figure 4-23).

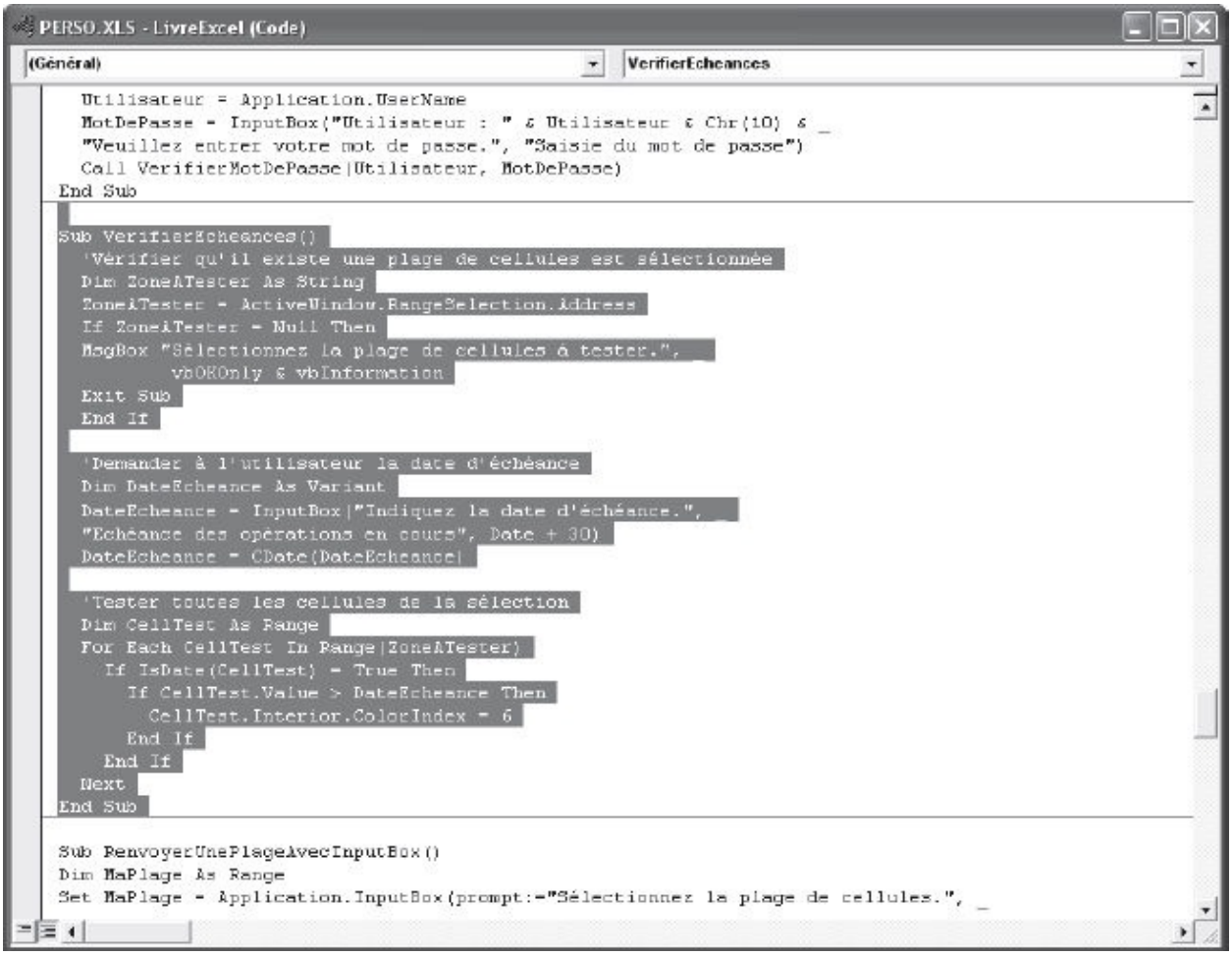

PERSO.XLS - LivreExcel (Code)
(Général)
VerifierEcheances
Utilisateur = Application.UserName
MotDePasse = InputBox("Utilisateur : " & Utilisateur & Chr(10) & _
"Veuillez entrer votre mot de passe.", "Saisie du mot de passe")
Call VerifierMotDePasse(Utilisateur, MotDePasse)
End Sub
Sub VerifierEcheances()
'Vérifier qu'il existe une plage de cellules est sélectionnée
Dim ZoneATester As String
ZoneATester = ActiveWindow.RangeSelection.Address
If ZoneATester = Null Then
MsgBox "Sélectionnez la plage de cellules à tester.", _
vbOKOnly & vbInformation
Exit Sub
End If
'Demander à l'utilisateur la date d'échéance
Dim DateEcheance As Variant
DateEcheance = InputBox("Indiquez la date d'échéance.", _
"Echéance des opérations en cours", Date + 30)
DateEcheance = CDate(DateEcheance)
'Tester toutes les cellules de la sélection
Dim CellTest As Range
For Each CellTest In Range(ZoneATester)
If IsDate(CellTest) = True Then
If CellTest.Value > DateEcheance Then
CellTest.Interior.ColorIndex = 6
End If
End If
Next
End Sub
Sub RenvoyerUnePlageAvecInputBox()
Dim MaPlage As Range
Set MaPlage = Application.InputBox(prompt:="Sélectionnez la plage de cellules.", _

Figure 4-23 – *Vous pouvez sélectionner rapidement une procédure entière.*

- Vous pouvez aussi vous déplacer ou effectuer des sélections de blocs de code à l'aide du clavier. Pour atteindre la procédure précédente (resp. suivante), tapez le raccourci clavier Ctrl+Pg. Préc (resp. Ctrl+Pg. Suiv). Pour étendre la sélection, utilisez les mêmes combinaisons de touches, auxquelles vous ajouterez la touche Maj.

Astuce

En substituant les touches fléchées haut et bas aux touches Pg. Préc. et Pg. Suiv. dans les combinaisons mentionnées, vous atteignez la première ligne de code sous la déclaration de procédure (`Sub`, `Function` ou `Property`) plutôt que la déclaration de procédure elle-même.

Recherche et remplacement de texte

Vous recherchez du texte dans la fenêtre Code comme vous le feriez dans un logiciel de traitement de texte tel que Word. Procédez comme suit :

1. À partir d'une fenêtre Code, sélectionnez la commande Rechercher du menu Édition, tapez le raccourci clavier Ctrl+F ou cliquez sur le bouton Rechercher de la barre d'outils Standard de Visual Basic Editor.

   La boîte de dialogue représentée à la figure 4-24 s'affiche.

Figure 4-24 – *La boîte de dialogue Rechercher de Visual Basic Editor propose des options communes à l'essentiel des traitements de texte.*

2. Dans la zone Rechercher, saisissez le texte voulu.
3. Dans la zone Dans, sélectionnez la portée de la recherche :
   - Procédure en cours. La recherche ne porte que sur le texte de la procédure en cours, c'est-à-dire celle dans laquelle se trouve le curseur ;
   - Module en cours. La recherche porte sur l'ensemble du module en cours, c'est-à-dire le texte de la fenêtre Code active ;
   - Projet en cours. La recherche porte sur l'ensemble des modules du projet, que leurs fenêtres de code respectives soient ouvertes ou non ;
   - Texte sélectionné. La recherche porte sur la plage de texte sélectionnée dans la fenêtre active.
4. Dans la liste déroulante, sélectionnez le sens dans lequel s'effectuera la recherche par rapport à l'emplacement du curseur : vers le haut, vers le bas ou dans les deux sens.
5. Cochez éventuellement les cases des options de recherche :
   - Mot entier. Le mot est recherché en tant que mot entier et non en tant que suite de caractères faisant partie d'un autre mot ;
   - Respecter la casse. La recherche porte sur le texte dont la casse (majuscules ou minuscules) est identique à celle du texte saisi dans la

zone Rechercher ;

– Critères spéciaux. Lorsque cette case est cochée, vous pouvez utiliser les caractères génériques (?, *, #, [`listedecar`] et [!`listedecar`]) dans la zone Rechercher.

6. Cliquez sur le bouton Suivant.

   Si la recherche aboutit, le texte trouvé s'affiche en surbrillance dans la fenêtre Code. Si la recherche porte sur le projet entier et si la chaîne est trouvée dans un autre module, la fenêtre de Code de ce module est ouverte. Pour atteindre une autre occurrence du texte, cliquez de nouveau sur le bouton Suivant.

   Si le texte recherché n'est pas trouvé, un message vous l'indique.

   Pour fermer la boîte de dialogue, cliquez sur le bouton Annuler ou sur sa case de fermeture.

Astuce

Une pression sur la touche F3 ou la commande Suivant du menu Édition relance la dernière recherche effectuée sans ouvrir la boîte de dialogue Rechercher.

Pour remplacer du texte, procédez comme suit :

1. Sélectionnez la commande Remplacer du menu Édition ou tapez le raccourci clavier Ctrl+H. Si la boîte de dialogue Rechercher est ouverte, cliquez sur le bouton Remplacer. La boîte de dialogue de la figure 4-25 s'affiche.

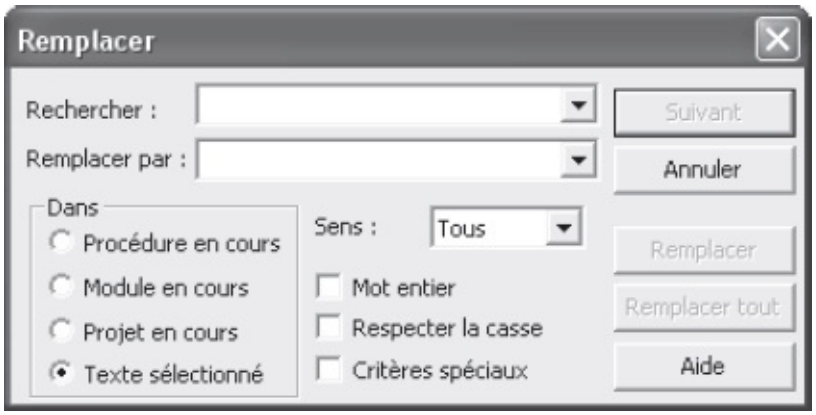


Figure 4-25 – *La boîte de dialogue Remplacer.*

2. Indiquez le texte à rechercher et définissez l'étendue, le sens et les options de recherche. Dans la zone Remplacer par, saisissez le texte de

remplacement.

3. Si vous souhaitez effectuer le remplacement sur toutes les occurrences du texte recherché, cliquez sur Remplacer tout.

   Une boîte de dialogue vous indique le nombre de remplacements effectués.

4. Pour visualiser les occurrences de texte trouvées avant d'effectuer le remplacement, cliquez sur le bouton Suivant pour lancer la recherche. Lorsque le texte est trouvé, il apparaît en surbrillance dans la fenêtre Code.

   Pour remplacer l'occurrence sélectionnée, cliquez sur le bouton Remplacer. Le texte est remplacé et la recherche se poursuit. Pour poursuivre sans remplacement, cliquez sur le bouton Suivant.

Affichage de plusieurs fenêtres Code

Lorsque vous développerez dans Visual Basic Editor, vous vous déplacerez souvent entre les codes de différents éléments de votre projet, voire échangerez des instructions d'une fenêtre à l'autre pour vous épargner une nouvelle saisie. Pour passer d'une fenêtre Code à l'autre, ouvrez le menu Fenêtre de la barre de menus de Visual Basic Editor et sélectionnez celle que vous souhaitez passer au premier plan.

Pour afficher simultanément plusieurs fenêtres Code, procédez comme suit :

1. Placez-vous dans l'Explorateur de projet et ouvrez les fenêtres de code concernées.
2. Sélectionnez Mosaïque horizontale (voir figure 4-26) ou Mosaïque verticale (voir figure 4-27) du menu Fenêtre.

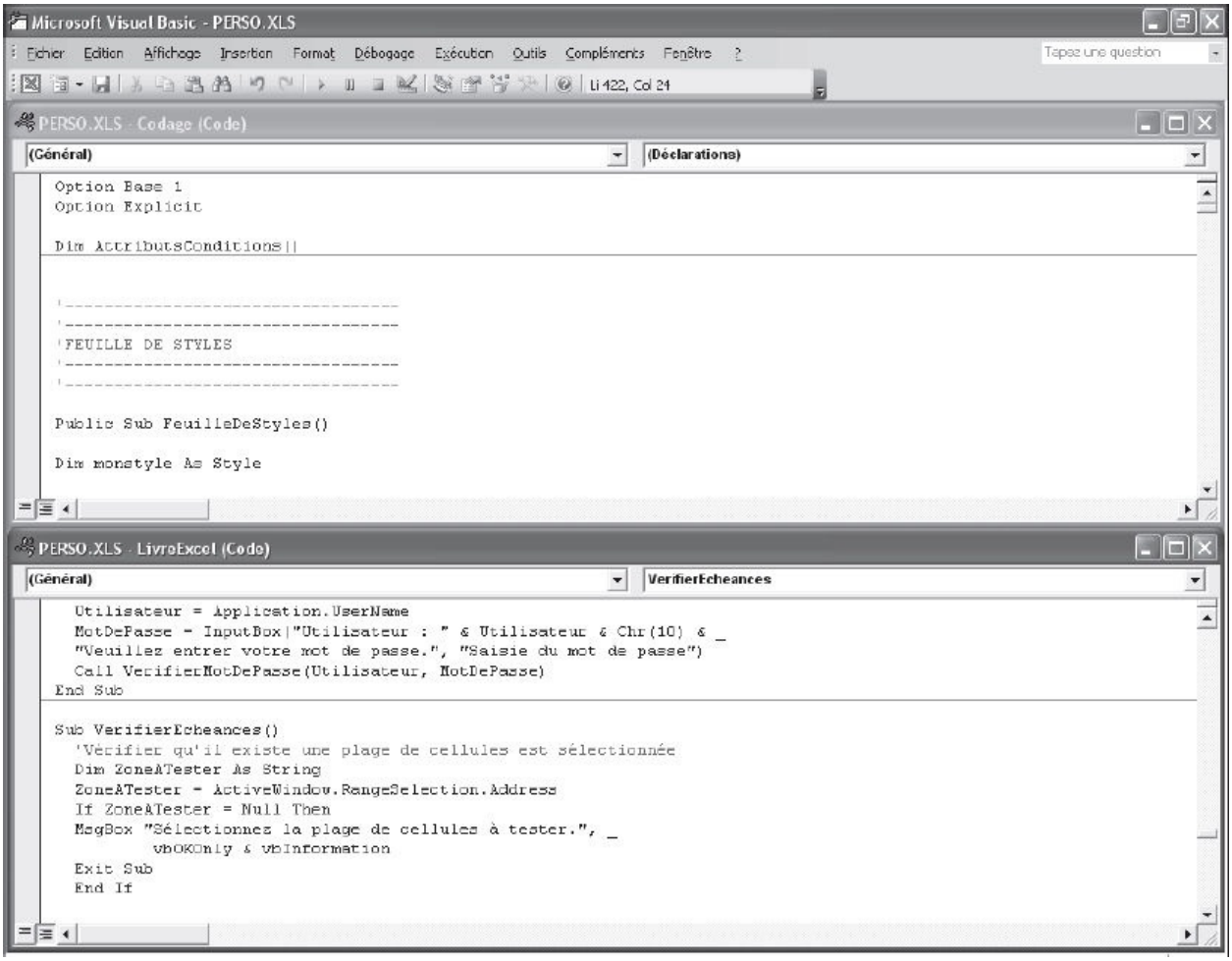

Microsoft Visual Basic - PERSO.XLS
Fichier Edition Affichage Insertion Format Débogage Exécution Outils Compléments Fenêtre ?
Tapez une question
Li 422, Col 24
PERSO.XLS - Codage (Code)
(Général)
(Déclarations)
Option Base 1
Option Explicit
Dim AttributsConditions()
'FEUILLE DE STYLES
Public Sub FeuilleDeStyles()
Dim monstyle As Style
PERSO.XLS - LivreExcel (Code)
(Général)
VerifierEcheances
Utilisateur = Application.UserName
MotDePasse = InputBox("Utilisateur : " & Utilisateur & Chr(10) & _
"Veuillez entrer votre mot de passe.", "Saisie du mot de passe")
Call VerifierMotDePasse(Utilisateur, MotDePasse)
End Sub
Sub VerifierEcheances()
'Vérifier qu'il existe une plage de cellules est sélectionnée
Dim ZoneATester As String
ZoneATester = ActiveWindow.RangeSelection.Address
If ZoneATester = Null Then
MsgBox "Sélectionnez la plage de cellules à tester.", _
vbOKOnly & vbInformation
Exit Sub
End If

Figure 4-26 – *Les fenêtres Code organisées en mosaïque horizontale.*

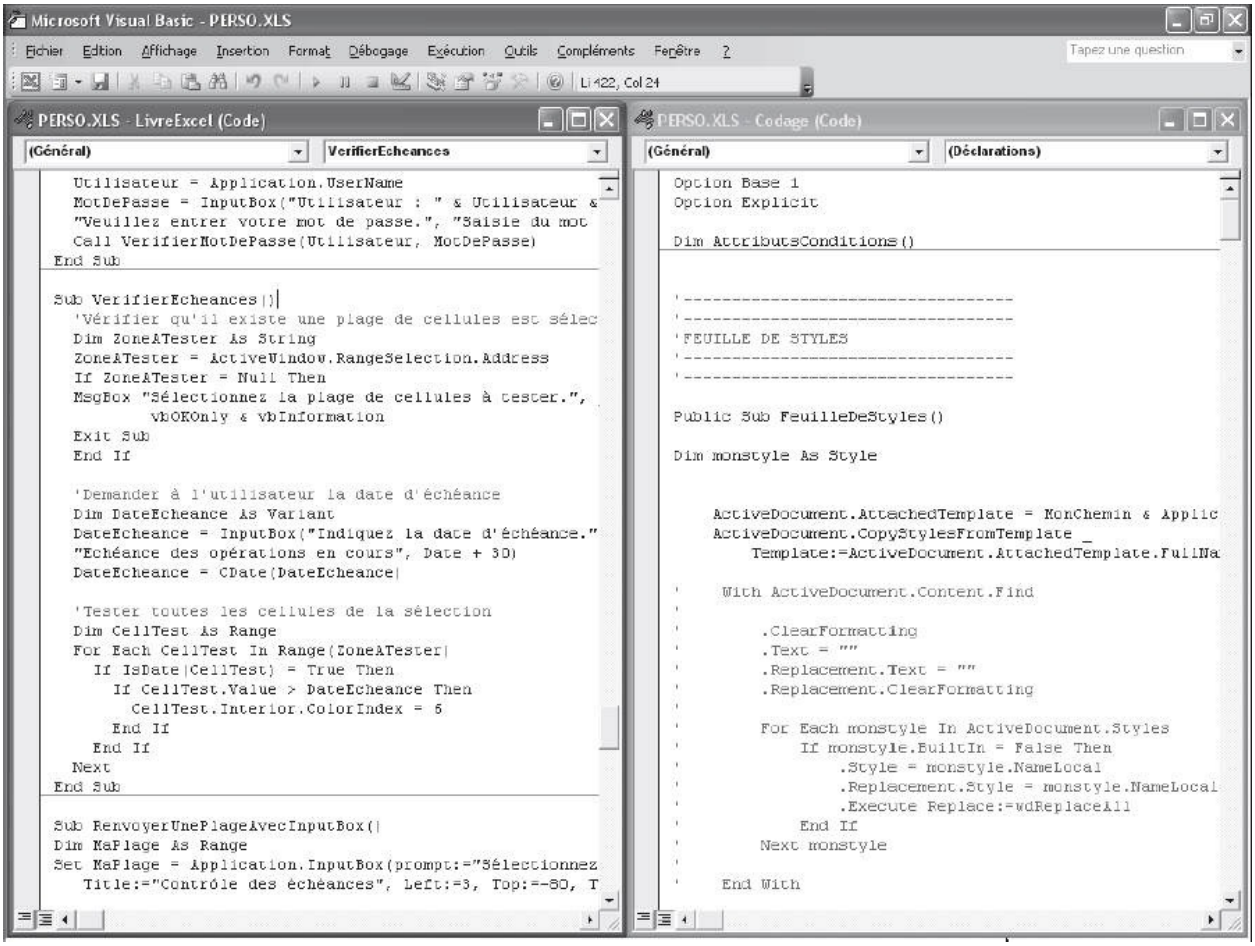


Figure 4-27 – *Les fenêtres Code organisées en mosaïque verticale.*

Il est aussi possible de séparer une fenêtre Code en deux volets, afin d'afficher simultanément des sections non contiguës d'un même module. Vous pouvez ainsi copier, coller ou déplacer du code d'une section à l'autre du module. Procédez comme suit :

1. Placez le curseur sur la barre de fractionnement située dans le haut de la barre de défilement verticale de la fenêtre Code.
2. Lorsque le curseur se transforme en une double flèche, faites glisser la barre de fractionnement vers le bas, jusqu'à atteindre la délimitation souhaitée pour les deux volets. Relâchez le bouton de la souris.

Les deux volets étant autonomes (voir figure 4-28), vous pouvez modifier l'affichage de l'un ou l'autre à l'aide des barres de défilement verticales. Les listes Objet et Procédure s'appliquent à la fenêtre active, c'est-à-dire celle dans laquelle se trouve le point d'insertion.

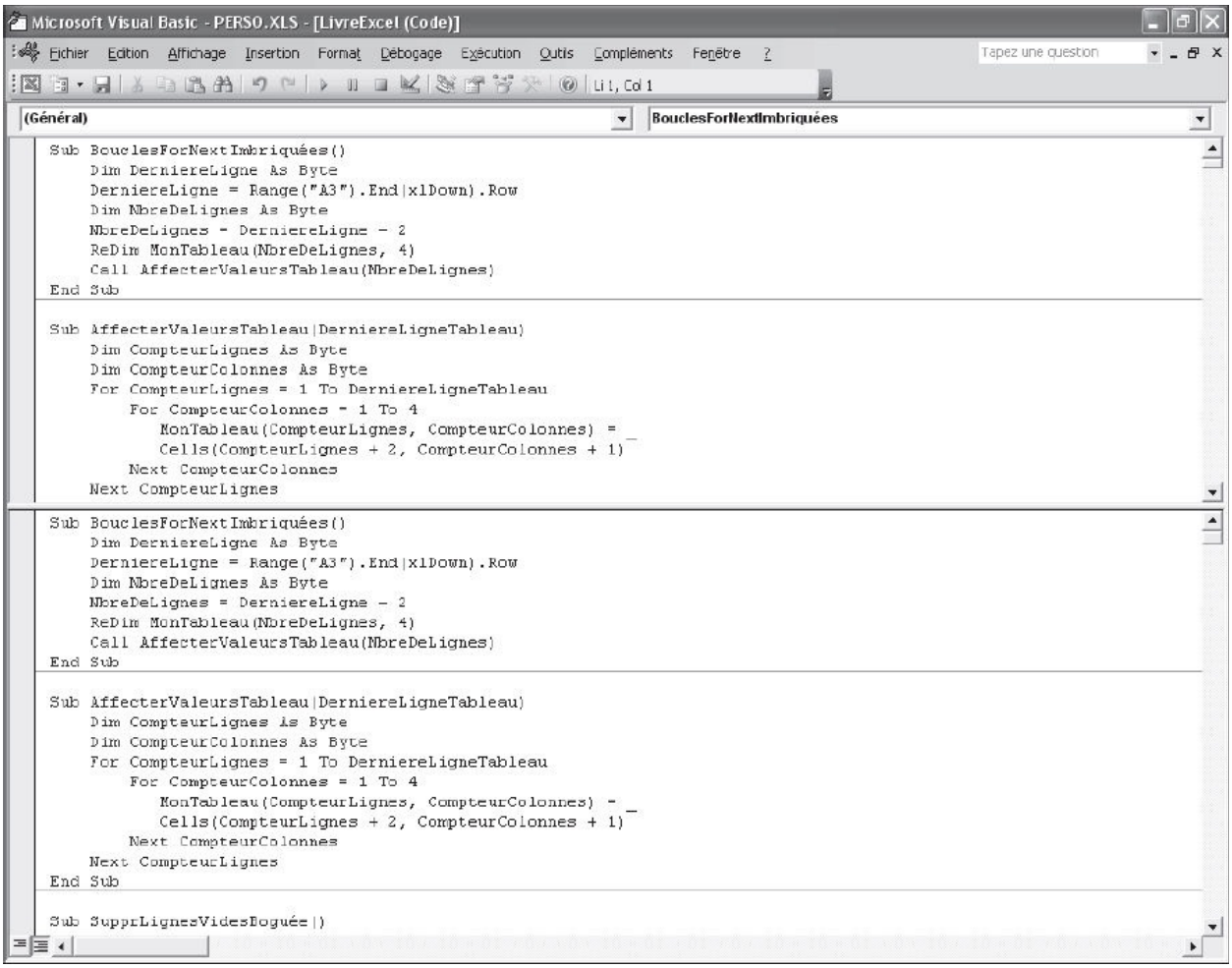


Figure 4-28 – *Le fractionnement de la fenêtre Code permet d'afficher simultanément différentes sections de code d'un même module.*

Pour annuler le fractionnement de la fenêtre Code, double-cliquez sur la barre de fractionnement ou faites-la glisser vers le haut de la fenêtre.

Options d'affichage

L'affichage dans la fenêtre Code de Visual Basic Editor est personnalisable. Cette section présente les options de paramétrage de la fenêtre ; celles du code sont présentées au chapitres 5.

Par défaut, la fenêtre Code affiche toutes les procédures du module, séparées par des traits gris (voir figure 4-29).

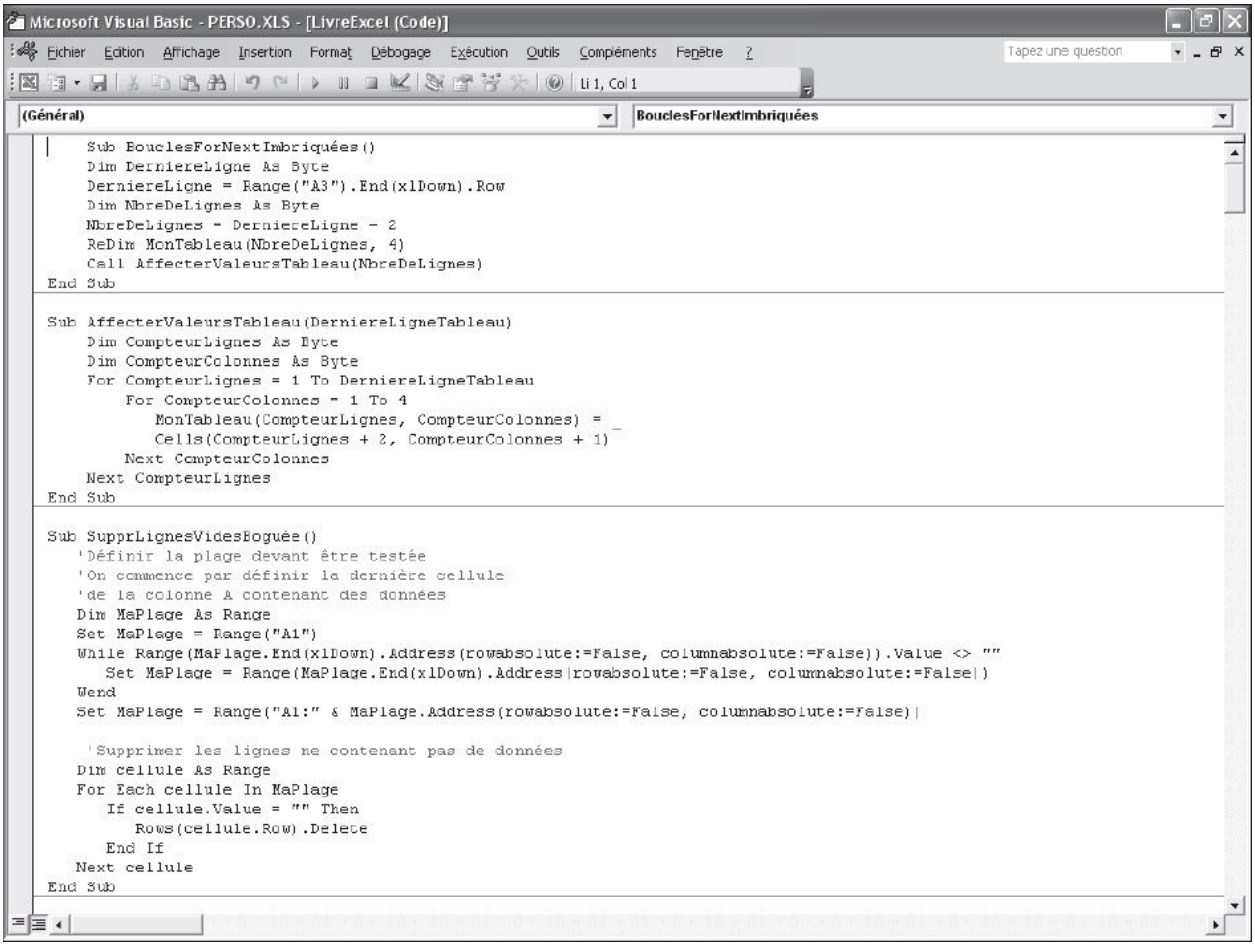

Microsoft Visual Basic - PERSO.XLS - [LivreExcel (Code)]
Fichier Edition Affichage Insertion Format Débogage Exécution Outils Compléments Fenêtre ?
Tapez une question
Li 1, Col 1
(Général)
BouclesForNextImbriquées
Sub BouclesForNextImbriquées()
Dim DerniereLigne As Byte
DerniereLigne = Range("A3").End(xlDown).Row
Dim NbreDeLignes As Byte
NbreDeLignes = DerniereLigne - 2
ReDim MonTableau(NbreDeLignes, 4)
Call AffecterValeursTableau(NbreDeLignes)
End Sub
Sub AffecterValeursTableau(DerniereLigneTableau)
Dim CompteurLignes As Byte
Dim CompteurColonnes As Byte
For CompteurLignes = 1 To DerniereLigneTableau
For CompteurColonnes = 1 To 4
MonTableau(CompteurLignes, CompteurColonnes) =
Cells(CompteurLignes + 2, CompteurColonnes + 1)
Next CompteurColonnes
Next CompteurLignes
End Sub
Sub SupprLignesVidesBoguée()
'Définir la plage devant être testée
'On commence par définir la dernière cellule
'de la colonne A contenant des données
Dim MaPlage As Range
Set MaPlage = Range("A1")
While Range(MaPlage.End(xlDown).Address(rowabsolute:=False, columnabsolute:=False)).Value <> ""
Set MaPlage = Range(MaPlage.End(xlDown).Address(rowabsolute:=False, columnabsolute:=False))
Wend
Set MaPlage = Range("A1:" & MaPlage.Address(rowabsolute:=False, columnabsolute:=False))
'Supprimer les lignes ne contenant pas de données
Dim cellule As Range
For Each cellule In MaPlage
If cellule.Value = "" Then
Rows(cellule.Row).Delete
End If
Next cellule
End Sub

Figure 4-29 – *L'ensemble des procédures du code s'affiche.*

Vous préférerez peut-être n'afficher qu'une procédure dans la fenêtre Code. Procédez comme suit :

1. Choisissez la commande Options du menu Outils. Sélectionnez l'onglet Éditeur de la boîte de dialogue qui s'affiche (voir figure 4-30).
2. Cochez ou décochez les cases de la zone Paramètres de la fenêtre, en fonction du type d'affichage souhaité :
   - Glisser-déplacer pour l'édition de texte ;
   - Affichage complet du module par défaut. Décochez cette case si vous souhaitez n'afficher qu'une procédure (voir figure 4-31). Utilisez alors la liste déroulante Procédure pour sélectionner celle que vous voulez ;
   - Séparation des procédures. Dans le cas d'un affichage complet du module, cette option détermine si un séparateur sera affiché ou non entre les différentes procédures.
3. Cliquez ensuite sur OK pour valider les options choisies.

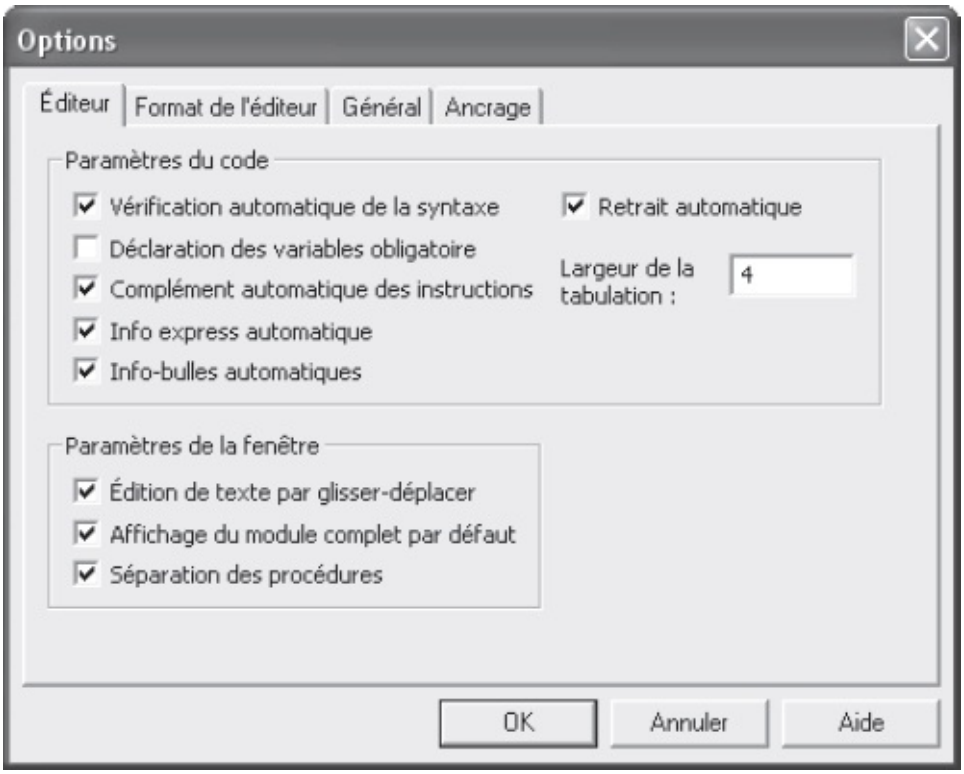


Figure 4-30 – *La zone Paramètres de la fenêtre sert à définir l'affichage des procédures.*

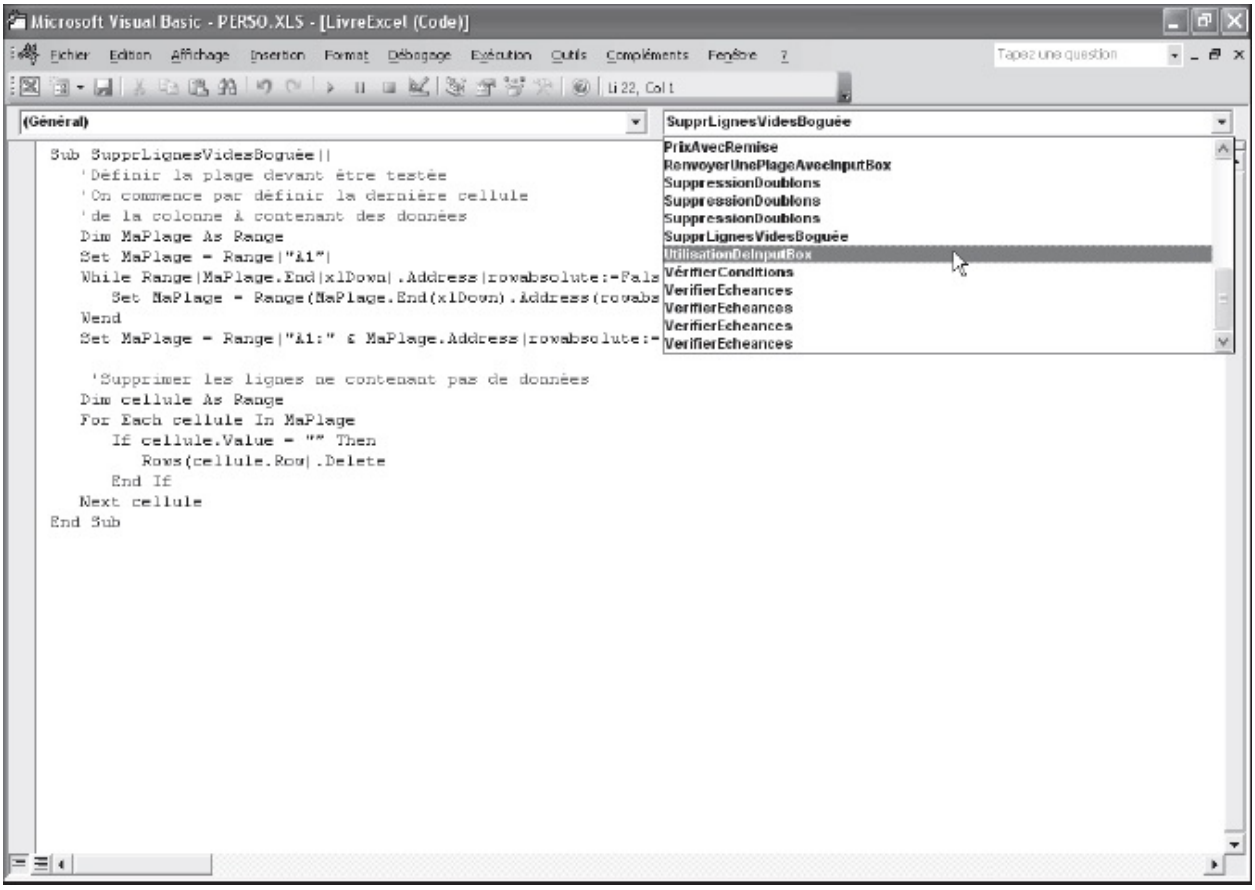


Figure 4-31 – *Vous pouvez choisir de n'afficher qu'une procédure dans la fenêtre Code.*

La fenêtre Propriétés

La fenêtre Propriétés concerne divers éléments d'un projet. Vous pouvez y visualiser et modifier les propriétés d'un classeur ou d'une feuille de calcul, d'une feuille UserForm, d'un contrôle, d'une classe, d'un projet ou d'un module. Vous l'utiliserez essentiellement lors du développement de feuilles pour vos projets.

Info

Vous découvrirez les propriétés essentielles des contrôles que l'on place sur les feuilles *UserForm* et apprendrez à les modifier aux chapitres 13 et 15.

Les propriétés varient en fonction de l'élément sélectionné. Ainsi, celles d'un contrôle en définissent l'aspect et le comportement (sa position, sa taille, le fait que l'utilisateur peut ou non y apporter des modifications, etc.), tandis que

celles d'un projet en définissent simplement le nom. La figure 4-32 présente la fenêtre Propriétés d'un contrôle `CheckBox` (case à cocher), et la figure 4-33 celle d'un projet.

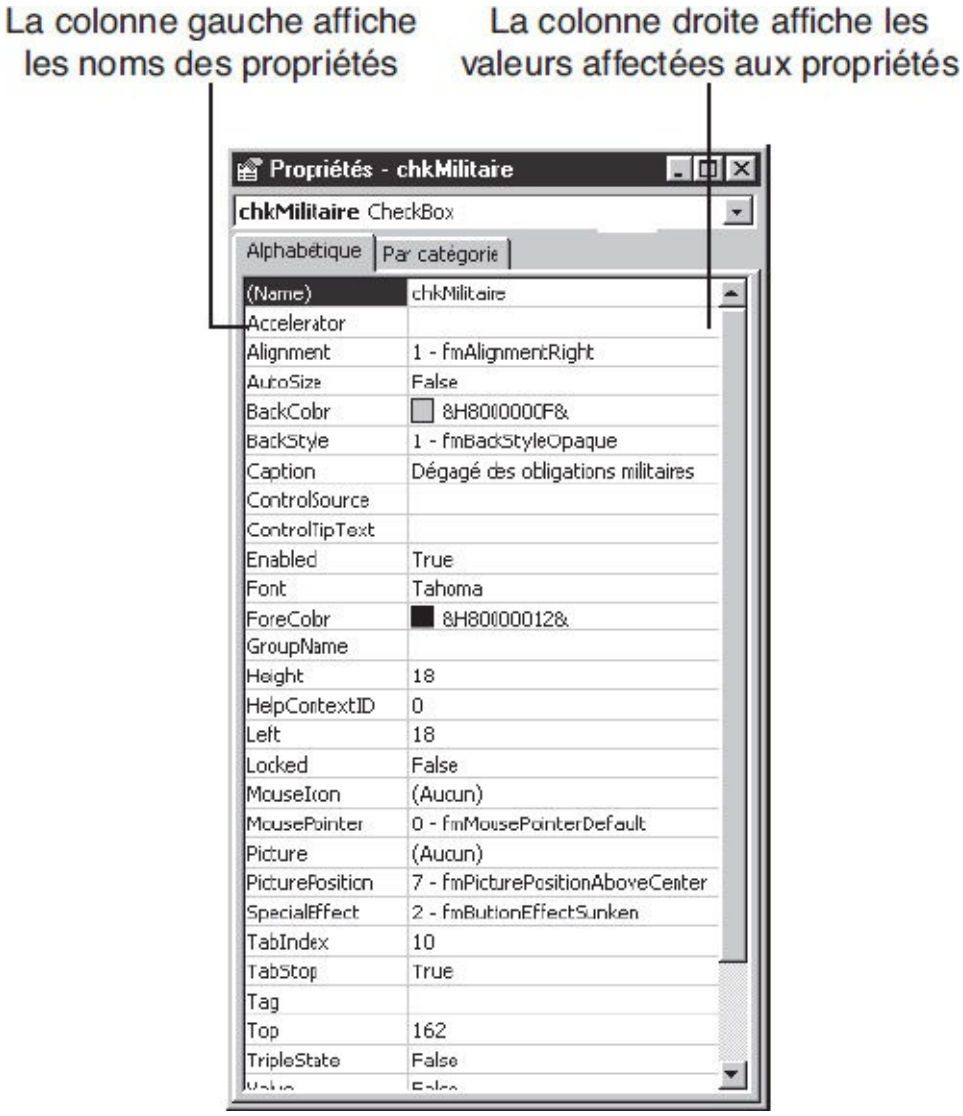


Figure 4-32 – *La fenêtre Propriétés d'un contrôle CheckBox.*

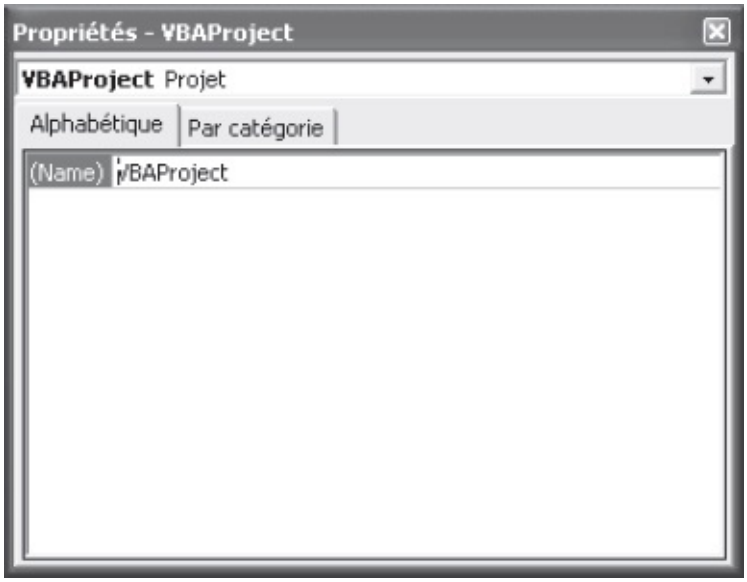


Figure 4-33 – *La fenêtre Propriétés d'un projet.*

Afficher et masquer la fenêtre Propriétés

La fenêtre Propriétés s'affiche à partir de l'Explorateur de projet ou à partir d'une fenêtre UserForm. Procédez comme suit :

1. Sélectionnez soit un module dans l'Explorateur de projet, soit un contrôle ou une forme dans la fenêtre UserForm.

Info

Lorsque plusieurs contrôles sont sélectionnés sur une feuille, la fenêtre Propriétés affiche leurs propriétés communes.

2. Dans le menu Affichage, sélectionnez la commande Propriétés, ou cliquez sur le bouton Propriétés de la barre d'outils Standard, ou encore tapez le raccourci clavier F4.
3. Lorsque vous sélectionnez un autre module dans l'Explorateur de projet ou un autre contrôle dans la fenêtre UserForm, les propriétés affichées sont automatiquement mises à jour.

Pour masquer la fenêtre Propriétés, cliquez sur la case de fermeture située à l'extrémité droite de la barre de titre ou cliquez-droit dans la fenêtre et, dans le menu contextuel qui s'affiche, sélectionnez la commande Masquer.

Naviguer dans la fenêtre Propriétés

Déroulez la liste des contrôles de la feuille active et sélectionnez celui dont vous souhaitez afficher les propriétés (voir figure 4-34).

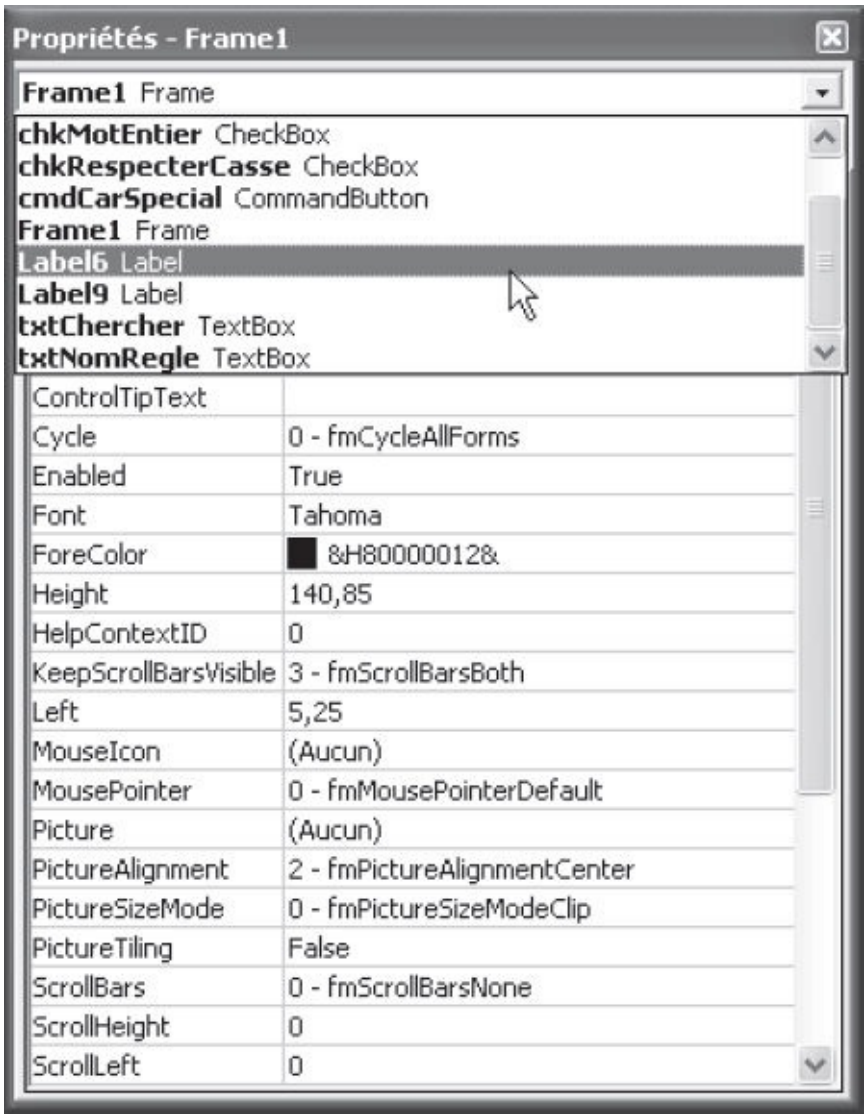


Figure 4-34 – *Sélectionnez l'objet dont vous souhaitez afficher les propriétés.*

La fenêtre Propriétés présente deux onglets déterminant le type d'affichage des propriétés :

- Alphabétique. Toutes les propriétés recensées pour l'élément sélectionné sont affichées par ordre alphabétique (voir figure 4-32) ;
- Par catégorie. Les propriétés de l'objet sélectionné sont regroupées par catégorie et, à l'intérieur de chacune, par ordre alphabétique (voir figure 4-

35). Les catégories apparaissent en gras. À l'instar des dossiers de l'Explorateur, les propriétés qui y sont recensées peuvent être affichées ou masquées lorsque vous cliquez sur les signes plus (+) ou moins (–).

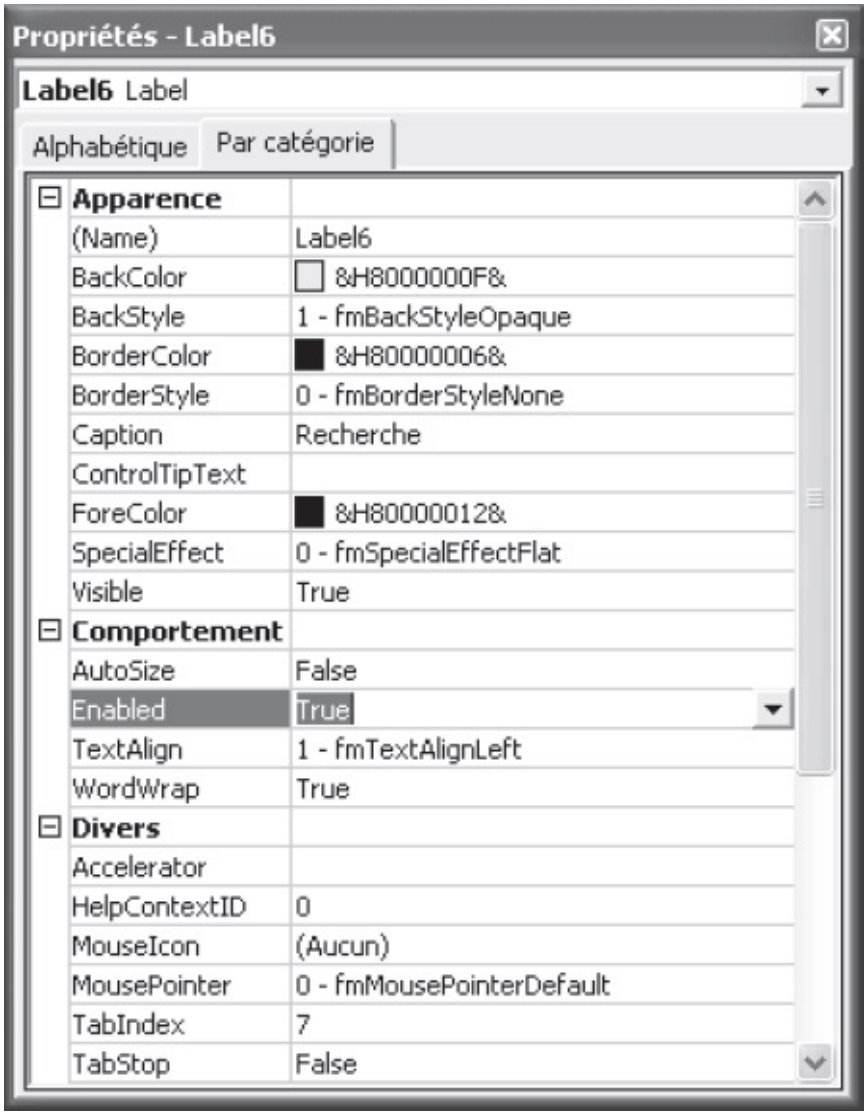


Figure 4-35 – *L'affichage par catégorie permet d'accéder rapidement aux propriétés voulues.*

Les catégories de propriétés essentielles sont :

- Apparence : propriétés relatives à l'aspect d'un objet (couleur de fond, intitulé, texte de son info-bulle, visibilité, etc.) ;
- Comportement : propriétés déterminant les réactions du contrôle aux actions de l'utilisateur (s'il est ou non modifiable, longueur maximale de saisie d'une chaîne, etc.) ;

- Défilement : propriétés concernant le déplacement dans un contrôle (présence ou non de barres de défilement, dimension et position relative des barres de défilement) ;
- Divers : propriétés variées, telles que le nom permettant d'appeler ce contrôle dans le code ;
- Emplacement : propriétés concernant la taille d'un contrôle et sa position sur la feuille ;
- Image : si une image est associée à un contrôle, vous trouverez ici ses propriétés spécifiques (source, position relative) ;
- Police : police utilisée pour l'intitulé du contrôle. Dans Office, la police utilisée par défaut pour les contrôles est le Tahoma.

Modifier une propriété

Comme nous l'avons vu au chapitres 1, des valeurs de différents types peuvent être affectées à une propriété :

- chaîne de caractères ;
- valeur numérique ;
- constante ;
- valeur booléenne.

Pour modifier la valeur d'une propriété, procédez comme suit :

1. Sélectionnez cette propriété. Elle apparaît en surbrillance. Appuyez sur la touche Tab pour sélectionner la valeur en cours pour cette propriété.
2. Affectez ensuite la valeur voulue. La démarche varie selon le type de valeur :
   - Si la propriété accepte une chaîne de caractères ou une valeur numérique, saisissez directement la valeur souhaitée au clavier ;
   - S'il s'agit d'une valeur booléenne ou d'une constante, une flèche apparaît dans la cellule de valeur. Cliquez dessus pour dérouler la liste des valeurs possibles pour la propriété et sélectionnez celle qui convient (voir figure 4-36). Vous pouvez aussi utiliser les touches fléchées haut et bas pour vous déplacer d'une valeur à l'autre ;

   - Si la propriété attend une chaîne indiquant un emplacement de fichier, un bouton servant à parcourir les fichiers disponibles apparaît.

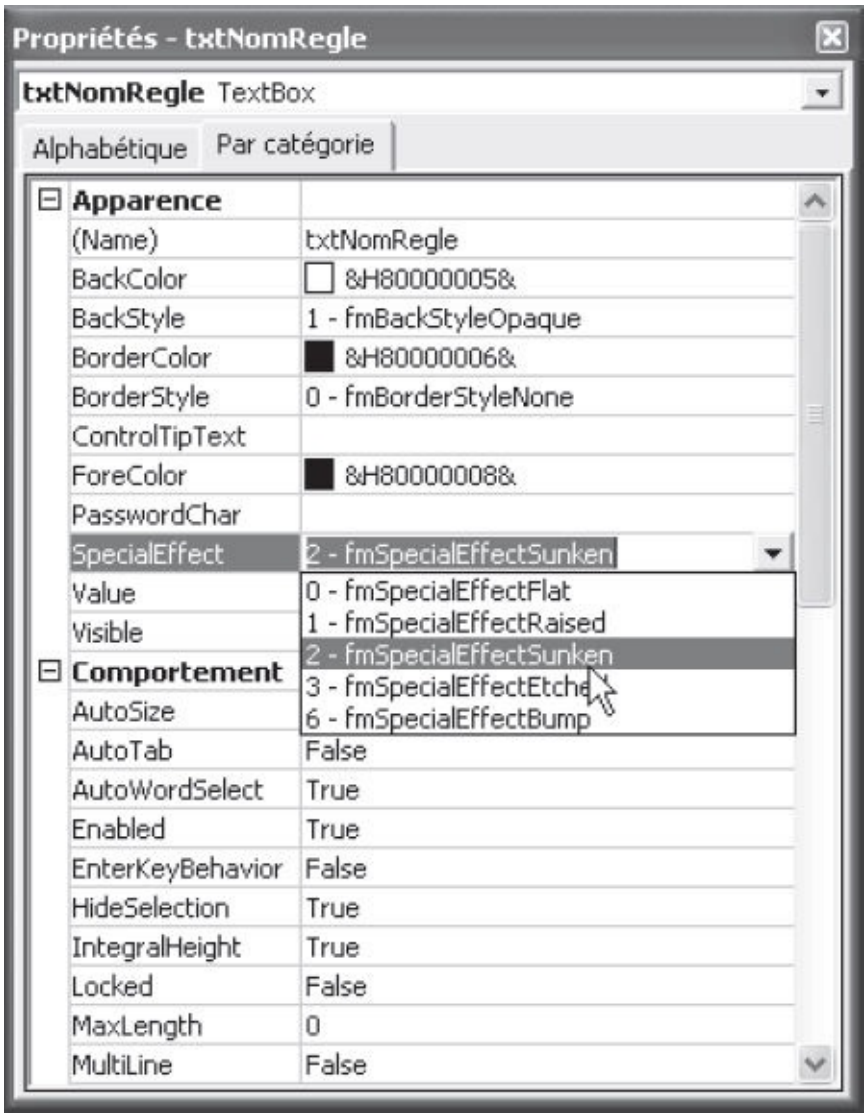


Figure 4-36 – *Lorsqu'une propriété accepte des valeurs prédéterminées, celles-ci sont recensées dans une liste déroulante.*

Astuce

Dans le cas des propriétés acceptant un booléen ou une constante, vous pouvez passer d'une valeur à la suivante en double-cliquant simplement dans la case de valeur.

Les barres d'outils

Visual Basic Editor contient quatre barres d'outils, présentées dans les figures suivantes.

Figure 4-37 – *La barre d'outils Standard propose les fonctions les plus communes de Visual Basic Editor.*

Figure 4-38 – *La barre d'outils Édition sert à obtenir de l'aide, mettre en forme le texte et s'y déplacer.*

Figure 4-39 – *La barre d'outils Débogage sert à tester le comportement d'un programme.*

Figure 4-40 – *La barre d'outils UserForm propose des outils pour organiser les contrôles sur une feuille.*

Cette section présente la barre d'outils Standard. Vous découvrirez les autres barres au fur et à mesure de la lecture de cet ouvrage.

Afficher, masquer et déplacer une barre d'outils

Cliquez-droit sur une barre d'outils ou sur la barre de menus de la fenêtre Visual Basic Editor, ou encore sélectionnez la commande Barre d'outils du menu Affichage, puis choisissez la barre à afficher (cochée) ou masquer (décochée).

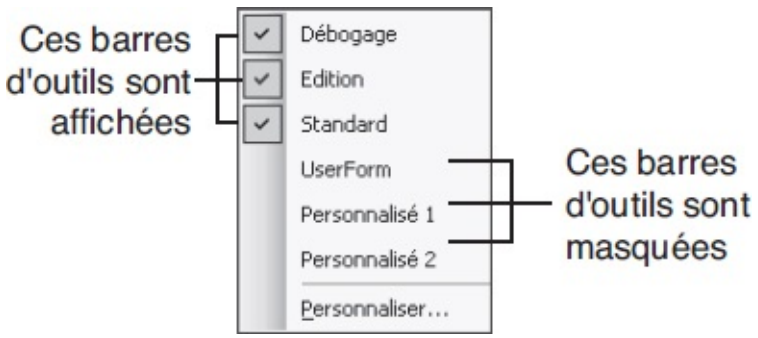


Figure 4-41 – *Le menu contextuel Barre d'outils.*

Il est possible de cliquer-déplacer une barre d'outils jusqu'à atteindre l'emplacement voulu, matérialisé par un contour grisé.

Astuce

Pour obtenir des informations contextuelles sur les boutons d'une barre, sélectionnez la commande Options du menu Outils et, dans l'onglet Général, cochez l'option Afficher les info-bulles.

La barre d'outils Standard

La barre d'outils Standard offre un accès rapide aux commandes les plus usitées. On y retrouve des commandes communes à l'essentiel des applications, telles que Enregistrer ou Couper. Le tableau 4-2 présente les boutons de cette barre d'outils.

Tableau 4-2. Les icônes de la barre d'outils Standard

Bouton	Nom	Description
	Afficher Microsoft Excel	Bascule entre l'application hôte et le document Visual Basic actif (raccourci clavier : Alt+F11).
	Insertion	En cliquant sur la flèche, vous ouvrez un menu permettant d'insérer l'un des objets suivants dans le projet actif : UserForm (feuille) Module de classe Module Procédure Le bouton représente le dernier objet ajouté (par défaut la feuille UserForm).
	Enregistrer <*Document hôte*>	Enregistre le document hôte, y compris le projet et tous ses composants – feuilles et modules (raccourci clavier : Ctrl+S).
	Couper	Supprime le texte sélectionné dans la fenêtre Code ou le contrôle sur la feuille active et le place dans le Presse-papiers (raccourci clavier : Ctrl+X).
	Copier	Copie dans le Presse-papiers le texte sélectionné dans la fenêtre Code ou le contrôle sur la feuille active (raccourci clavier : Ctrl+C).
	Coller	Insère le contenu du Presse-papiers à l'emplacement courant. Ce bouton n'est accessible que si ce contenu est compatible avec la fenêtre active – fenêtre Code pour du texte et fenêtre UserForm pour un contrôle (raccourci clavier : Ctrl+V).
	Rechercher	Recherche une chaîne dans une fenêtre Code – voir la section « La fenêtre Code », plus haut dans ce chapitre (raccourci clavier : Ctrl+F).

	Annuler	Chaque clic sur ce bouton annule la dernière modification effectuée (raccourci clavier : Ctrl+Z).
	Répéter	Chaque clic sur ce bouton rétablit la dernière action annulée. Pour qu'il soit accessible, il faut qu'une action ait été annulée et qu'aucune modification n'ait eu lieu depuis l'annulation.
	Exécuter Sub/UserForm ou Exécuter la macro	Exécute une application fonction de la fenêtre active : • Fenêtre Code : la procédure en cours est exécutée. • Fenêtre UserForm : la feuille active est exécutée. • Autre : ouvre la boîte de dialogue Macros du projet actif, à partir de laquelle vous pouvez exécuter la macro de votre choix.
	Arrêt	Interrompt l'exécution en cours et bascule en mode Arrêt. En mode Arrêt, vous pouvez relancer l'exécution d'une procédure, réinitialiser un projet, tester un programme, etc. (raccourci clavier : Ctrl+Pause).
	Réinitialiser	Réinitialise le projet.
	Mode Création/Quitter le mode Création	Active ou désactive le mode Création. Contrairement au mode Exécution, le code d'un projet n'y est pas exécuté. Cela correspond à la période de développement de votre projet – création d'une feuille UserForm ou écriture de code. En mode Création, vous pouvez placer des contrôles sur les feuilles de calcul et leur affecter du code.
	Explorateur de projet	Affiche la fenêtre Explorateur de projet (raccourci clavier : Ctrl+R).
	Fenêtre Propriétés	Affiche la fenêtre Propriétés de l'objet sélectionné (raccourci clavier : F4).
	Explorateur d'objets	Affiche la fenêtre Explorateur d'objets (raccourci clavier : F2).
	Boîte à outils	Affiche la boîte à outils. Accessible uniquement si une fenêtre UserForm est active.
	Assistant Office	Ouvre la fenêtre de l'Assistant Office, dans laquelle vous êtes invité à saisir le sujet sur lequel vous souhaitez obtenir de l'aide (raccourci clavier : F1).

Paramétrer Visual Basic Editor

L'affichage de Visual Basic Editor est personnalisable. Testez les différentes options de façon à trouver les paramétrages qui vous conviennent le mieux.

Comme la plupart des fenêtres Windows, celles de Visual Basic Editor sont déplaçables par cliquer-glisser sur la barre de titre de chacune, et redimensionnables en tirant leurs bords. Vous pouvez aussi choisir d'ancrer une ou plusieurs fenêtres à l'une des bordures de la fenêtre de Visual Basic Editor.

Définition

Une fenêtre est dite ancrée lorsqu'elle est fixée à la bordure d'une autre (celle de l'application ou une autre fenêtre elle-même ancrée).

Une fenêtre ancrée demeure au premier plan. Si vous maximisez une fenêtre dans Visual Basic Editor, celle-ci viendra épouser la fenêtre ancrée, sans en recouvrir l'espace. Vous pouvez ainsi tirer pleinement parti de l'espace de Visual Basic Editor. À la figure 4-42, nous avons ancré l'Explorateur de projet et maximisé une fenêtre Code.

Les fenêtres ancrables sont les suivantes :

- Exécution ;
- Variables locales ;
- Espions ;
- Explorateur de projet ;
- Propriétés ;

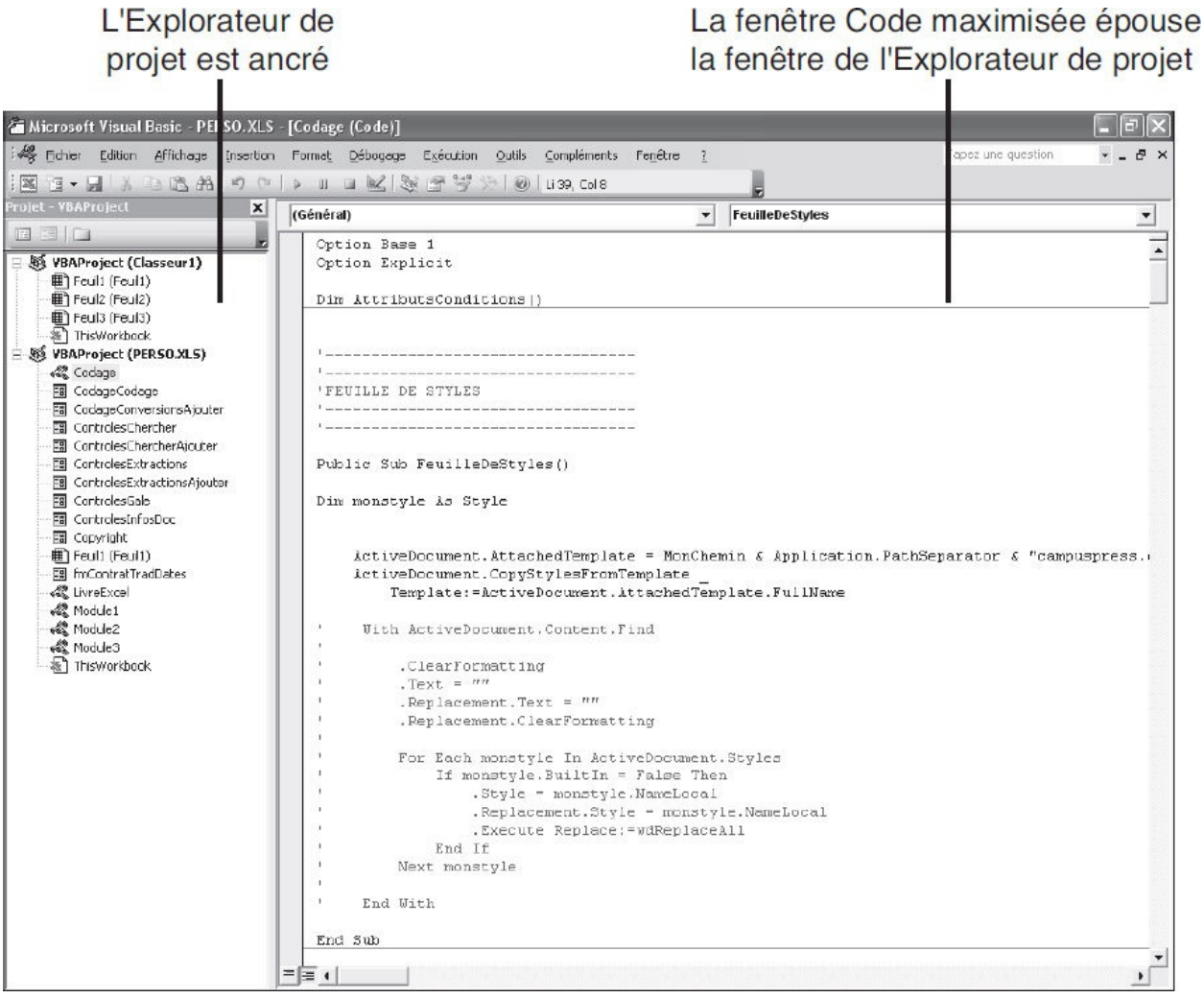


Figure 4-42 – *Ancrer l'Explorateur de projet facilite l'accès aux éléments de vos projets.*

- Explorateur d'objets.

Lorsqu'une fenêtre est ancrée, elle se place automatiquement sur l'une des bordures de la fenêtre de Visual Basic Editor quand vous la déplacez. Si vous souhaitez pouvoir la déplacer n'importe où dans la fenêtre de Visual Basic Editor, désactivez-en l'ancrage.

Pour activer ou désactiver l'ancrage des fenêtres de Visual Basic Editor, choisissez l'une des méthodes suivantes :

- Cliquez-droit dans la fenêtre de votre choix. Un menu contextuel s'affiche, fonction de la fenêtre choisie. Sélectionner/désélectionner la commande Ancrable (figure 4-43) active ou désactive l'ancrage de la fenêtre.
- Sélectionnez la commande Options du menu Outils et placez-vous sur l'onglet Ancrage (voir figure 4-44). Cochez les cases des fenêtres que vous souhaitez ancrer et décochez les autres. Cliquez sur OK.

Info

Si l'accès aux commandes de Visual Basic Editor ne convient pas à votre façon de travailler, personnalisez les menus et les barres d'outils en y ajoutant/supprimant des commandes et des boutons. Vous pouvez aussi créer une nouvelle barre d'outils ou un nouveau menu dans lesquels vous placerez les commandes de votre choix. Pour connaître les procédures de personnalisation, reportez-vous au chapitres 11.

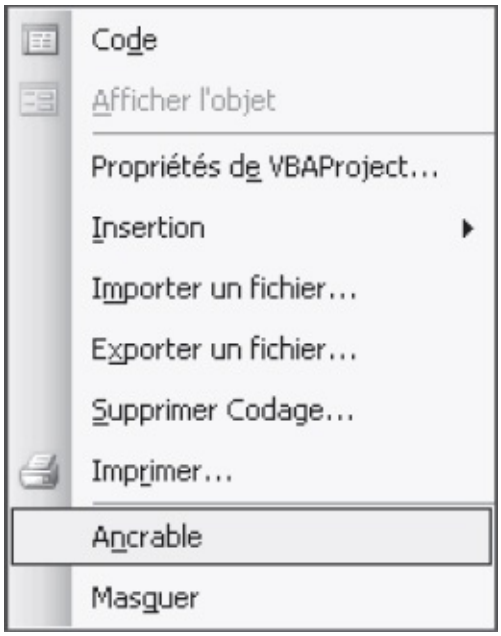


Figure 4-43 – *Les menus contextuels des fenêtres de Visual Basic Editor possèdent une commande Ancrable.*

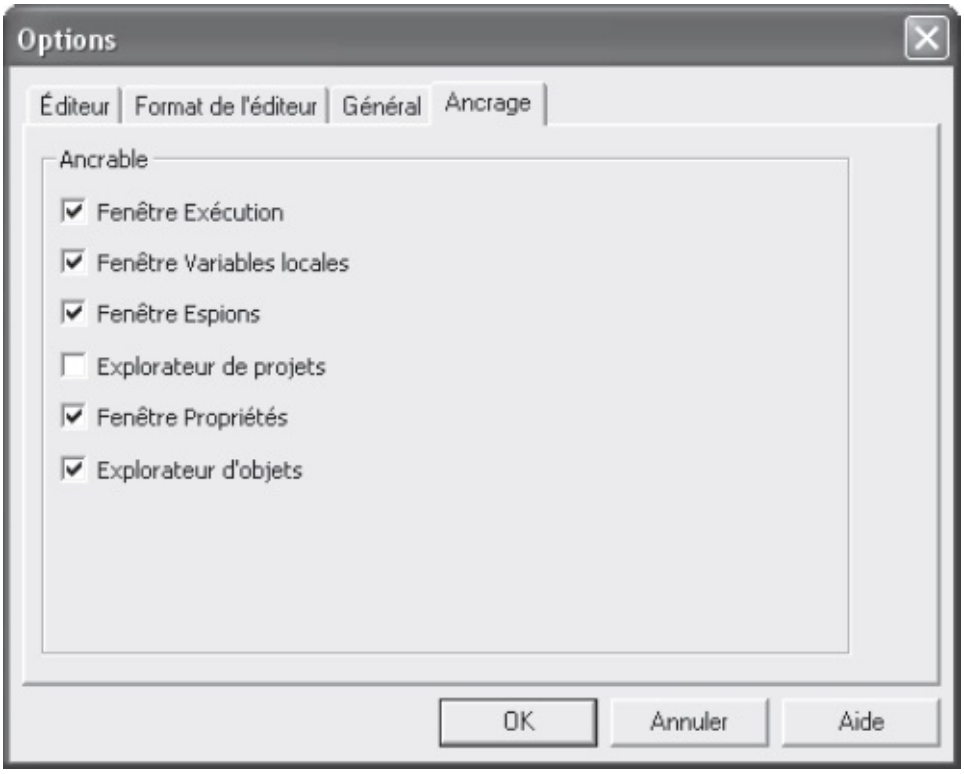


Figure 4-44 – *La boîte de dialogue Options permet de modifier les options d'ancrage de plusieurs fenêtres simultanément.*

DEUXIÈME PARTIE

Programmer en Visual Basic

5

Développer dans Visual Basic Editor

Les chapitres précédents vous ont permis d'acquérir les concepts essentiels et de découvrir l'environnement de développement Visual Basic Editor. Avec ce chapitre, nous entrons de plain-pied dans la programmation VBA.

Vous apprendrez à distinguer les composants essentiels et à déterminer les besoins de votre projet. Vous serez ainsi à même de le structurer de façon cohérente, lui assurant efficacité et lisibilité. Gardez à l'esprit que les projets sont attachés à une application hôte, ici Excel. Vous devez donc ouvrir Visual Basic Editor à partir d'Excel. Le classeur auquel est affecté (ou auquel vous souhaitez affecter) votre projet doit aussi être ouvert.

Structure des programmes Visual Basic

Les projets VBA sont constitués d'objets distincts, dont l'ensemble constitue un programme entier.

Les modules

Comme vous l'avez vu en découvrant l'Explorateur de projet, on distingue les modules standards ou modules de code, les modules de classe et les feuilles. Autrement dit, les différents composants du code d'un projet VBA sont structurés et distingués selon leur type. Ces éléments interagissent et s'appellent pour constituer un programme complet.

Le code décrivant l'interface d'un programme et celui affecté aux différents événements qui peuvent toucher cette interface (un clic de souris sur un bouton OK, par exemple) sont stockés dans un fichier UserForm. Pour chaque feuille d'un projet, il existe un objet UserForm accessible dans le dossier Feuilles de l'Explorateur de projet. Le projet de la figure 5-1 contient dix feuilles.

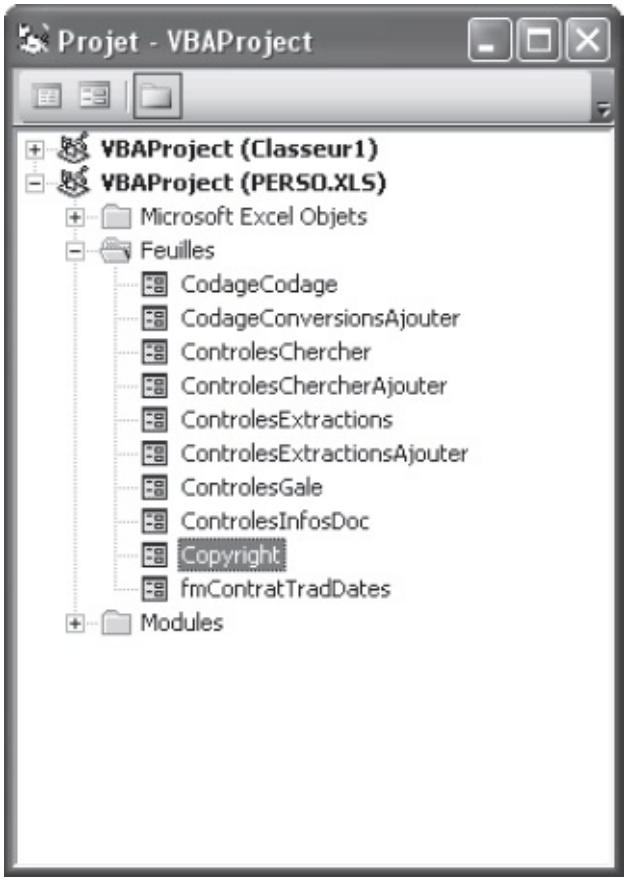


Figure 5-1 – *À chaque feuille d'un projet est affecté un fichier dans le dossier Feuilles.*

Le code standard se trouve dans des modules de code, stockés dans le dossier Modules, tandis que le code décrivant les objets développés pour votre projet est stocké dans le dossier Modules de classe.

Les procédures

À l'intérieur d'un même module, le code est structuré en *procédures*. Une procédure est une séquence d'instructions s'exécutant en tant qu'entité. Cette décomposition rend le code plus performant et plus lisible.

Par exemple, lorsqu'un projet VBA ouvre une boîte de dialogue (une feuille), pour chaque événement déclenché par l'utilisateur, l'application vérifie s'il existe une procédure (une unité de code) affectée à cet événement dans le module correspondant. Si tel est le cas, la procédure est exécutée.

C'est l'ensemble des procédures d'un projet, avec leurs interactions, qui forme un programme complet. Par exemple, l'événement clic de souris sur un bouton

OK d'une boîte de dialogue peut déclencher une procédure qui récupère et traite l'ensemble des informations contenues dans cette boîte de dialogue (Cette case est-elle cochée ? Quel est le texte saisi dans cette zone de texte ?, etc.). Cette procédure peut ensuite *appeler* (ou *invoquer*) une autre procédure stockée dans le Module de code du projet et lui *passer* les informations ainsi traitées. On parle alors de *procédure appelante* et de *procédure appelée* ; cette dernière effectuera les tâches pour lesquelles elle a été écrite en exploitant les données fournies par la première. Les informations transmises sont les *arguments passés*.

Définition

On qualifie de procédure événementielle (ou d'événement) une procédure déclenchée par un événement spécifique, tel qu'un clic de souris ou la frappe d'une touche clavier, par opposition aux procédures standards d'un module de code, indépendantes de toute interaction utilisateur.

Info

Dans les modules UserForm, les procédures sont prédéterminées ; il en existe une pour chaque événement susceptible d'affecter un contrôle. Dans les modules de code, c'est vous qui déterminez les différentes procédures. Vous pourriez notamment décider de n'en écrire qu'une seule contenant tout le programme. Il est cependant conseillé de structurer le code en procédures distinctes. La lecture et le débogage de vos programmes en seront considérablement améliorés.

Nous vous conseillons de structurer vos codes en petites procédures effectuant chacune un traitement spécifique. Le programme principal consiste alors en une procédure qui appelle les autres. Examinez le programme suivant :

```
Sub CalculPaieNette()
   NomRepr = InputBox("Entrez le nom du représentant : ")
   Call VerifierNomRepresentant(NomRepr)
   SalaireRepr = QuelSalaire(NomRepr)
   ChiffreRepr = InputBox("Entrez le chiffre d'affaires réalisé : ")
   Prime = CalculPrime(ChiffreRepr)
   MsgBox "La paie nette sera de " & CalculPaie(SalaireRepr, Prime)
End Sub
```

Cette procédure est composée de six instructions, dont quatre font appel à d'autres procédures pour exécuter des tâches spécifiques. `VerifierNomRepresentant` est appelée afin de vérifier que le nom indiqué est valide ; si ce n'est pas le cas, elle prendra en charge la résolution du problème ou appellera une autre procédure conçue dans ce but. `QuelSalaire` et `CalculPrime` sont ensuite appelées afin de renvoyer le salaire et la prime du représentant. Enfin, `CalculPaie` est appelée pour renvoyer la paie du représentant, qui s'affiche dans une boîte de dialogue.

Les techniques d'appels de procédures sont traitées en détail plus loin dans ce chapitre.

Cette façon de procéder présente plusieurs avantages :

- Les programmes sont plus faciles à lire et, chaque procédure prenant en charge une tâche spécifique, il est plus aisé d'en comprendre le fonctionnement ;
- En cas de bogue du programme, il est plus facile d'isoler la procédure coupable et de corriger le problème ;
- Le code est ainsi réutilisable. Si une tâche a été isolée dans une petite procédure, elle peut être exploitée par différents programmes ;

  Dans l'exemple précédent, `VerifierNomRepresentant` est appelée pour contrôler la validité du nom entré par l'utilisateur. Cette procédure peut être appelée par d'autres programmes, sans qu'il soit nécessaire de la réécrire.

Les instructions

Une procédure est composée d'instructions, chacune exécutant une tâche précise, qui peut être évidente ou invisible pour l'utilisateur et destinée à effectuer des traitements propres au projet.

Une instruction Visual Basic est composée de *mots-clés* du langage, de constantes et de variables. Ces éléments peuvent être combinés pour former une *expression* qui vérifie des données ou effectue une tâche.

Par exemple, `Mavar = Workbooks.Count` est une instruction. Elle combine la variable `Mavar` et les mots-clés `=`, `Workbooks` et `Count`. Pour affecter à la variable `Mavar` une valeur égale au nombre de classeurs ouverts, on associe cette variable à l'expression `Workbooks.Count` à l'aide de l'opérateur arithmétique `=` (la propriété `Workbooks` renvoie la collection d'objets `Workbooks` représentant l'ensemble des classeurs ouverts et la propriété `Count` renvoie le nombre d'objets de la collection spécifiée).

Définition

Un mot-clé est un mot ou un symbole reconnu comme élément du langage Visual Basic. Il peut s'agir d'une fonction, d'une propriété, d'une méthode ou encore d'un opérateur arithmétique.

Rappel

Une constante est un élément nommé affecté à une valeur qui, contrairement à une variable, ne change pas durant l'exécution du programme. Le nom de la constante est utilisé à la place de la valeur qui lui est affectée. Un programme exploite des constantes propres à l'application ou définit ses propres valeurs à l'aide de l'instruction `Const` (voir chapitres 6).

Vous pouvez par exemple définir une constante que vous nommerez TVA et à laquelle vous affecterez la valeur 0,186. Chaque fois que vous aurez besoin de cette valeur dans votre programme, il vous suffira d'utiliser le nom de la constante qui lui est affectée. Ainsi, l'expression :

```
PrixHorsTaxe * TVA
```

sera équivalente à :

```
PrixHorsTaxe * 0.186
```

Rappel

Les variables sont définies par un nom autre qu'un mot-clé du langage ; elles servent à stocker des informations modifiées au cours de l'exécution du programme.

On distingue trois types d'instructions :

- Les instructions de déclaration : invisibles pour l'utilisateur, elles servent à nommer une variable, une constante ou une procédure. Le nom attribué à un élément dans l'instruction de déclaration sera ensuite utilisé pour invoquer cet élément tout au long du projet. Une instruction de déclaration peut aussi déterminer le *type* de l'élément déclaré.

  `Sub MaProcédure()` est un exemple d'instruction de déclaration utilisée pour nommer la procédure `MaProcédure`. `Dim Mavar As String` est aussi une instruction de déclaration. L'instruction `Dim` sert à nommer la variable `MaVar`, tandis que `As String` en spécifie le type (une chaîne de caractères).

- Les instructions d'affectation : elles affectent une valeur ou une expression à une variable, à une constante ou encore à une propriété – et contiennent donc toujours l'opérateur `=`. L'exécution de ce type d'instructions peut être visible comme invisible pour l'utilisateur.

  `MaVar = 5` et `MaVar = Workbooks.Count` sont des exemples d'instructions d'affectation, attribuant respectivement une valeur et une expression à la variable `MaVar`.

  `Let` est l'instruction d'affectation de Visual Basic. Ainsi l'instruction `MaVar = 5` peut aussi être écrite sous la forme `Let MaVar = 5`. Cependant, elle est facultative et généralement omise.

  `ActiveSheet.Range("C1").Font.Name = "Arial"` est une instruction d'affectation visible pour l'utilisateur puisqu'elle applique la police Arial à la cellule C1 de la feuille Excel active – la valeur "Arial" est affectée à la propriété `Name` de l'objet `Font` de cette cellule.

- Les instructions exécutables : elles accomplissent des actions (exécution d'une méthode, d'une fonction). Elles comprennent également les instructions de contrôle (traitées au chapitres 7).

`MsgBox "Quel est le nombre de classeurs ouverts ?", vbOKOnly + vbInformation, "Bonne question"` est une instruction exécutable entraînant l'affichage de la boîte de dialogue représentée à la figure 5-2. L'instruction `MsgBox` est étudiée au chapitres 7.

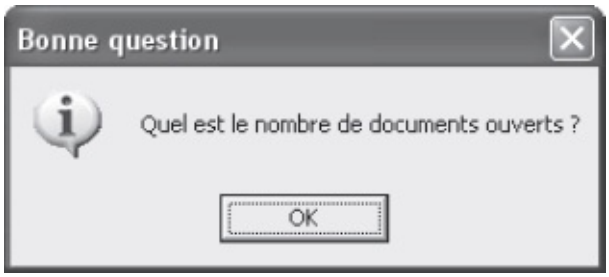


Figure 5-2 – *Les instructions MsgBox sont des instructions exécutables.*

La procédure suivante illustre l'utilisation conjointe des différents types d'instructions :

```
Sub AfficherNbreClasseursOuverts()
  Dim NbreClasseurs As Byte
  NbreClasseurs = Workbooks.Count
  MsgBox "Il y a actuellement " & NbreClasseurs & _
  " classeurs ouverts.", vbOKOnly + vbInformation, "Informations"
End Sub
```

Cette procédure affiche une boîte de dialogue indiquant le nombre de classeurs ouverts dans la session Excel active (voir figure 5-3).

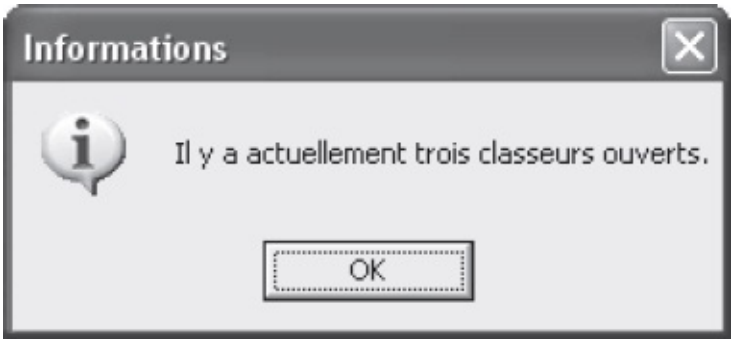


Figure 5-3 – *Dans une procédure, les instructions exécutables sont les seules visibles pour l'utilisateur.*

Les deux premières instructions sont des déclarations nommant successivement la procédure `AfficherNbreClasseursOuverts` et la variable `NbreClasseurs`. La valeur représentant le nombre de documents ouverts est ensuite stockée dans la variable `NbreClasseurs` dans l'instruction d'affectation de la ligne suivante. Enfin, l'instruction d'exécution `MsgBox` affiche une boîte de dialogue indiquant à

l'utilisateur le nombre de classeurs ouverts dans la session Excel. L'instruction de déclaration `End Sub` signale la fin de la procédure.

Les différents types de procédures

Une procédure peut exécuter une tâche visible par l'utilisateur, comme effectuer des tâches invisibles, internes au projet, et qui seront exploitées ultérieurement. On distingue différents types de procédures, en fonction du type de tâche qu'elles exécutent :

- `Sub` ;
- `Property` ;
- `Function`.

Procédures Sub

Une procédure `Sub` (ou sous-routine) est une série d'*instructions* exécutant une tâche déterminée au sein du projet, sans renvoyer de valeur. Elle est structurée de la façon suivante :

```
Sub NomDeLaProcédure()
   Instructions
   …
End Sub
```

Les instructions `Sub` et `End Sub` déterminent le début et la fin de la procédure. *`NomDeLaProcédure`* doit respecter les règles d'affectation de noms de Visual Basic présentées dans la note suivante. Ce nom est utilisé pour invoquer la procédure à partir d'une autre. Dans le cas d'une macro, il s'agit du nom attribué dans la boîte de dialogue Macro ou Enregistrer une macro. Les *`Instructions`* déterminent ce qu'exécute la procédure.

Attention

Les noms de procédures, comme ceux des variables et des constantes, doivent obéir aux règles de Visual Basic :

- contenir au maximum 255 caractères ;
- commencer par une lettre ;
- ne pas comprendre d'espace et ne pas utiliser les caractères @ & $ # . ! ;
- ne pas être identique à un mot-clé du langage, pour éviter tout conflit.

L'instruction de déclaration `Sub` peut aussi contenir des arguments optionnels, selon la syntaxe suivante :

```
[Private|Public] [Static] Sub NomDeLaProcédure([Arguments])
   Instructions
   …
End Sub
```

`Private` ou `Public` indique si la procédure est *privée* ou *publique*. Une procédure publique peut être invoquée n'importe où dans le projet, y compris dans d'autres modules. Une procédure privée ne peut être invoquée qu'à l'intérieur du même module. On parle de *portée* de la procédure. Lorsque les mots-clés `Private` et `Public` sont omis dans la déclaration de la procédure, cette dernière est publique.

L'option `Static` indique que les variables de la procédure `Sub` conservent leurs valeurs entre les différents appels. Autrement dit, si une procédure `Static` est invoquée à plusieurs reprises lors de l'exécution d'un programme, les variables qui lui sont propres ont la valeur qui leur a été affectée lors de l'appel précédent.

`Arguments` représente les arguments (des valeurs séparées par des virgules) passés par la procédure appelante.

```
Sub MaProcédure(arg1, arg2)
```

Dans cet exemple, `MaProcédure` est déclarée comme nécessitant les arguments `arg1` et `arg2`. Autrement dit, la procédure appelante devra lui transmettre ces arguments, ou bien une erreur sera générée. Les appels de procédures et le passage d'arguments sont étudiés dans la section « Appel et sortie d'une procédure », plus loin dans ce chapitre.

Chacun des arguments répond à la syntaxe suivante :

```
[Optional] [ByVal|ByRef] [ParamArray] NomVariable [As type] [=ValeurParDéfaut]
```

Tableau 5-1. Déclaration d'arguments dans une instruction Sub

Élément	Description
`Optional`	Ce mot-clé indique que les arguments transmis sont facultatifs. Aucune erreur ne sera générée si la procédure appelante ne les passe pas. Si vous souhaitez déclarer des arguments facultatifs et d'autres obligatoires, vous devez d'abord déclarer ceux qui sont obligatoires. Dans l'instruction `Sub MaProcédure(arg1, arg2, Optional arg3, Optional arg4)`, les arguments `arg1` et `arg2` doivent obligatoirement être passés par la procédure appelante, tandis que `arg3` et `arg4` sont facultatifs.
	Ces mots-clés indiquent respectivement que l'argument est passé *par valeur* ou *par référence*. Lorsqu'un argument est passé par valeur, c'est le contenu de la variable, et non son adresse, qui est transmis à la procédure appelée ; cette valeur est exploitée par la procédure, mais ne peut être modifiée.

`ByVal` ou `ByRef`	Lorsqu'un argument est passé par référence, c'est son adresse qui est transmise. La procédure exploite alors la valeur de la variable, mais peut aussi la modifier.
	Par défaut (lorsque ces mots-clés sont omis), le passage d'un argument se fait par référence.
	Dans l'exemple suivant :
	`Sub MaProcédure(ByVal MaVar)`
	`  MaVar = MaVar * 100`
	`End Sub`
	la valeur de la variable passée `MaVar` est multipliée par 100 dans `MaProcédure`, mais `MaVar` étant passée par valeur, son contenu réel ne sera pas modifié.

`ParamArray`	Ce mot-clé indique que l'argument est un tableau Optional. L'argument déclaré avec ce mot-clé doit être en dernière position dans la liste des arguments et ne peut utiliser conjointement l'un des mots-clés précédemment décrits dans ce tableau.
`As` *`Type`*	Spécifie le type de l'argument passé à la procédure : `Byte`, `Boolean`, `Integer`, `Long`, `Currency`, `Single`, `Double`, `Date`, `String`, `Object`, `Variant`, ou un type défini par l'utilisateur. Une erreur est générée si la variable passée par la procédure appelante est incompatible avec le type déclaré. L'instruction de déclaration de `MaProcédure` suivante indique qu'un argument `Arg1` de type `String` (chaîne de caractères) est requis : `Sub MaProcédure(Arg1 As String)` Les différents types de données ainsi que la création de types personnalisés sont étudiés au chapitres 6.
= *`ValeurParDéfaut`*	Indique une valeur par défaut pour l'argument. Ce paramètre ne peut être utilisé que conjointement avec le mot-clé `Optional`. Il détermine une valeur par défaut, qui sera employée si l'argument n'est pas passé par la procédure appelante. Il peut s'agir d'une constante (numérique, booléenne ou de type chaîne de caractères) ou d'une expression constante. Une expression renvoyant une valeur variable ne peut être utilisée. Dans l'exemple suivant, si l'argument `Arg1` n'est pas passé par la procédure appelante, sa valeur sera la chaîne "Bonjour". `Sub MaProcédure(Arg1 = "Bonjour")`

Attention

Si, dans la déclaration d'une procédure `Sub`, vous indiquez des arguments facultatifs à l'aide du mot-clé `Optional` sans spécifier de valeur par défaut, la procédure devra être conçue pour s'exécuter sans faire appel à ces arguments lorsqu'ils ne sont pas transmis. Dans le cas contraire, une erreur sera générée.

Le programme suivant illustre l'utilisation du mot-clé `Static` dans une instruction `Sub` :

```
Dim MaVar
Sub ProcédureAppelante()
  MaVar = 2
  Call ProcédureStatic(MaVar)
  Call ProcédureStatic(MaVar)
```

```
  Call ProcédureStatic(MaVar)
End Sub


Static Sub ProcédureStatic(MaVar)
  Dim MaVarStatique
  MaVarStatique = MaVarStatique + MaVar
  MsgBox MaVarStatique
End Sub
```

`ProcédureAppelante` invoque à trois reprises `ProcédureStatic` à l'aide de l'instruction `Call`, en lui passant la variable `MaVar` dont la valeur est 2. À chaque appel, `ProcédureStatic` incrémente la variable `MaVarStatique` de la valeur de `MaVar` et affiche sa valeur dans une boîte de dialogue à l'aide de la fonction `MsgBox`. La procédure appelante reprend alors la main (et invoque à nouveau la procédure `ProcédureStatic`).

Ce programme affiche donc à trois reprises une boîte de dialogue dont le message est successivement 2, 4, puis 6. En effet, la procédure appelée étant déclarée `Static`, la variable locale `MaVarStatique` conserve sa valeur entre deux appels. Elle est incrémentée de 2 à chaque exécution de la procédure.

Définition

Une variable est dite locale lorsqu'elle est propre à une procédure, par opposition à une variable publique ou passée par la procédure appelante.

Si vous supprimez l'instruction `Static` de la déclaration de `ProcédureStatic`, la variable `MaVarStatique` ne conservera plus sa valeur entre les différents appels et le programme affichera à trois reprises une boîte de dialogue dont le message sera toujours 2.

Le programme suivant illustre le passage d'un argument par valeur :

```
Dim MaVar
Sub ProcédureAppelante()
  MaVar = 2
  Call ProcédureStatic(MaVar)
  Call ProcédureStatic(MaVar)
  Call ProcédureStatic(MaVar)
End Sub


Static Sub ProcédureStatic(ByVal MaVar)
  Dim MaVarStatique
  MaVarStatique = MaVarStatique + MaVar
  MsgBox MaVarStatique
  MaVar = 100
End Sub
```

Ce programme se comporte comme nous l'avons détaillé dans l'exemple précédent, mais `ProcédureStatic` affecte en plus la valeur 100 à `MaVar` avant de redonner la main à la procédure appelante. Néanmoins, le programme

continue d'afficher trois boîtes de dialogue dont les messages sont successivement 2, 4 et 6. En effet, l'instruction de déclaration de `ProcédureStatic` spécifie que l'argument `MaVar` est appelé par valeur (`ByVal`) et non par adresse. La valeur de `MaVar` est donc exploitée, mais ne peut être modifiée. `MaVar` retrouve donc sa valeur initiale dès que la procédure appelante reprend la main.

Supprimez le mot-clé `ByVal` dans l'instruction de déclaration de `ProcédureStatic`. L'argument `MaVar` est maintenant passé par adresse, et sa valeur peut donc être modifiée. Le programme affiche maintenant trois boîtes de dialogue dont les messages sont successivement 2, 102 et 202.

Procédures Function

Une procédure `Function` (ou fonction) est une série d'*instructions* exécutant une tâche déterminée au sein du projet et renvoyant une valeur ; cette procédure sera exploitée par d'autres procédures. Elle est structurée de la façon suivante :

```
Function NomDeLaProcédure(Arguments)
   Instructions
   …
   NomDeLaProcédure = Expression
   …
End Function
```

Les instructions `Function` et `End Function` déterminent le début et la fin de la procédure. *`NomDeLaProcédure`* doit respecter les règles de Visual Basic ; ce nom est utilisé pour invoquer la procédure à partir d'une autre.

Les *`Instructions`* déterminent ce qu'exécute la procédure.

`NomDeLaProcédure = Expression` affecte une valeur à la fonction. Cette instruction d'affectation peut apparaître à plusieurs reprises et n'importe où dans le code de la fonction. La valeur renvoyée est alors la dernière reçue lorsque la fonction prend fin.

L'instruction de déclaration `Function` peut aussi contenir des arguments facultatifs, selon la syntaxe suivante :

```
[Private|Public] [Static] Function NomDeLaProcédure ([Arguments]) [As Type]
   Instructions
   …
   NomDeLaProcédure = Expression
End Function
```

`As` *`Type`* précise le type de valeur renvoyé par la procédure. Il peut s'agir de l'un de ceux présentés au chapitres 6. S'il n'est pas précisé, la fonction renverra une valeur de type `Variant`. Comme nous l'avons déjà mentionné, il est recommandé de préciser cet argument afin de limiter la mémoire employée par le

programme.

Les mots-clés `Private` et `Public` indiquent si la procédure est privée ou publique. Le mot-clé `Static` indique que les variables locales de la procédure sont statiques, c'est-à-dire conservent leurs valeurs entre les appels. Pour un rappel de ces concepts, reportez-vous à la section précédente, « Procédures Sub ».

`Arguments` représente les arguments, séparés par des virgules, passés à `Function` par la procédure appelante.

```
Sub MaFonction(arg1, arg2)
```

Dans cet exemple, `MaFonction` est déclarée comme nécessitant les arguments `arg1` et `arg2`. Autrement dit, la procédure appelante devra lui passer ces arguments, ou bien une erreur sera générée. Les appels de procédures et le passage d'arguments sont étudiés dans la section « Appel et sortie d'une procédure », plus loin dans ce chapitre.

Chacun des arguments répond à la syntaxe suivante :

```
[Optional] [ByVal|ByRef] [ParamArray] NomVariable [As type] [=ValeurParDéfaut]
```

Cette syntaxe est la même que pour les arguments d'une instruction `Sub` ; reportez-vous au tableau 5-1.

La procédure suivante calcule la surface d'un cercle :

```
Function SurfaceCercle(Rayon As Long) As Long
  Const Pi = 3.14
  SurfaceCercle = Pi * Rayon * Rayon
End Function
```

La première ligne déclare la fonction en indiquant que l'argument `Rayon` est requis. La fonction ainsi que l'argument attendu sont de type `Long`. Une constante `Pi` est ensuite définie. À la troisième ligne, on affecte à la fonction une expression calculant la surface du cercle dont on a transmis le rayon. Enfin, l'instruction `End Function` signale la fin de la procédure.

Les fonctions sont faciles à appeler à partir d'autres procédures. Vous pouvez les utiliser comme n'importe quelle fonction intégrée de Visual Basic, c'est-à-dire en faisant apparaître dans une expression son nom suivi de la liste des arguments requis entre parenthèses.

Considérez le programme suivant :

```
Sub MaProcédure()
  Dim Rayon
  Rayon = 10
  MsgBox "La surface du cercle est de " & SurfaceCercle(Rayon) & " centimètres
  ➥ carré.", _
         vbOKOnly + vbInformation, "Appel de fonction"
End Sub
```

```
Function SurfaceCercle(Rayon)
  Const Pi = 3.14
  SurfaceCercle = Pi * Rayon * Rayon
End Function
```

La sous-routine `MaProcédure` déclare la variable `Rayon` et lui affecte la valeur `10`. Elle affiche ensuite une boîte de dialogue dont une partie du message fait appel à la fonction `SurfaceCercle` en lui passant l'argument `Rayon`. Cette dernière calcule donc la surface du cercle et rend la main à la procédure appelante, qui affiche la boîte de dialogue présentée à la figure 5-4.

Figure 5-4 – *L'instruction MsgBox fait appel à la fonction SurfaceCercle pour afficher la surface du cercle.*

Procédures Property

Une procédure `Property` (ou procédure de propriété) est une série d'*instructions* exécutant une tâche déterminée au sein du projet et manipulant des données de type Propriétés. Il existe trois types de procédures `Property` :

- **`Property Get`**. Elles renvoient la valeur d'une propriété qui sera ensuite exploitée par d'autres procédures.
- **`Property Let`**. Ces procédures définissent la valeur d'une propriété.
- **`Property Set`**. Elles établissent une référence entre un objet et une propriété.

> **Rappel**
> Une propriété est un attribut nommé d'un objet, définissant ses caractéristiques ou son état. Par exemple, la propriété `Address` d'un objet `Range` en renvoie l'adresse (A1, par exemple) et la propriété `ColorIndex` d'un objet `Font` (une police de caractères) en renvoie la couleur.

Procédures Property Get

Une procédure `Property Get` est structurée de la façon suivante :

```
Property Get NomDeLaProcédure()
```

```
    Instructions
    …
    NomDeLaProcédure = Expression
    …
End Property
```

Les instructions `Property Get` et `End Property` déterminent le début et la fin de la procédure. *`NomDeLaProcédure`* doit respecter les règles d'affectation de noms de Visual Basic ; il est utilisé pour invoquer la procédure à partir d'une autre.

Les *`Instructions`* définissent ce qu'exécute la procédure.

`NomDeLaProcédure = Expression` affecte une valeur à la procédure de propriété. Cette instruction d'affectation peut apparaître à plusieurs reprises et n'importe où dans le code de la procédure.

L'écriture d'une procédure `Property Get` se justifie lorsque la valeur de la propriété ne peut être renvoyée en une seule instruction Visual Basic – par exemple, lorsque l'on souhaite renvoyer sous forme de chaîne de caractères une propriété affectée à une constante.

La procédure de propriété suivante renvoie une chaîne de caractères représentant un commentaire, lequel est fonction de la valeur de la cellule qui lui est passée. Cette information est renvoyée par la propriété `Value` de l'objet `Cellule`.

```
 1: Property Get RenvoyerCommentaire(Cellule As Range) As String
 2:     Select Case Cellule.Value
 3:       Case Is < 10000
 4:          RenvoyerCommentaire = "Très mauvais"
 5:       Case 10000 To 20000
 6:          RenvoyerCommentaire = "Mauvais"
 7:       Case 20001 To 30000
 8:          RenvoyerCommentaire = "Correct"
 9:       Case 30001 To 40000
10:          RenvoyerCommentaire = "Bon"
11:       Case Is > 40000
12:          RenvoyerCommentaire = "Très bon"
13:     End Select
14: End Property
```

Attention

Ce listing est numéroté de façon à simplifier la présentation des différentes instructions de la procédure `RenvoyerCommentaire`. La présence de cette numérotation dans la procédure réelle générerait évidemment une erreur à l'exécution.

À la ligne 1, l'instruction `Property Get` déclare la procédure `RenvoyerCommentaire`, qui doit recevoir l'argument `Cellule` de type `Range` et qui renvoie une valeur de type `String`. Une instruction de contrôle `Select Case` est utilisée des lignes 2 à 13 pour tester la valeur renvoyée par l'expression `Cellule.Value`. Pour chacune des plages

de valeurs testées, une chaîne de caractères est affectée à la procédure de propriété.

Les procédures `Property Get` sont simples à appeler à partir d'autres procédures. Vous pouvez les utiliser dans n'importe quelle expression exploitant une valeur de propriété, en faisant apparaître le nom de la procédure suivi de la liste des éventuels arguments entre parenthèses.

Considérez les procédures suivantes :

```
Sub DefinirCommentaire()
    Dim LaCellule As Range
    For Each LaCellule In Selection
      LaCellule.AddComment (RenvoyerCommentaire(LaCellule))
    Next LaCellule
End Sub


Property Get RenvoyerCommentaire(Cellule) As String
    Select Case Cellule.Value
      Case Is < 10000
         RenvoyerCommentaire = "Très mauvais"
      Case 10000 To 20000
         RenvoyerCommentaire = "Mauvais"
      Case 20001 To 30000
         RenvoyerCommentaire = "Correct"
      Case 30001 To 40000
         RenvoyerCommentaire = "Bon"
      Case Is > 40000
         RenvoyerCommentaire = "Très bon"
    End Select
End Property
```

La sous-routine `DefinirCommentaire` appelle la procédure de propriété `RenvoyerCommentaire`. Une variable objet de type `Range` y est créée, tandis qu'une structure `For Each…Next` est utilisée pour effectuer un traitement sur chaque cellule de la sélection en cours dans la feuille de calcul active. Pour l'instant, n'essayez pas de comprendre ces instructions. Les variables et les structures de contrôle – telles que `For Each...Next` et `Select Case` – sont présentées en détail dans les deux chapitres suivants.

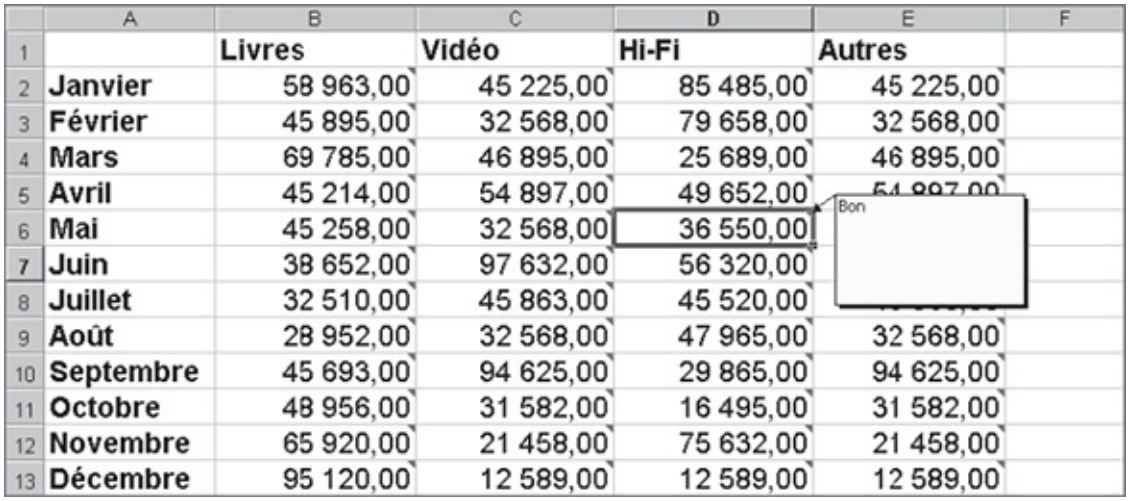
La méthode `AddComment` est appliquée à chaque cellule afin d'insérer un commentaire. Cette méthode s'utilise selon la syntaxe suivante :

```
Expression.AddComment(Texte)
```

où *`Expression`* renvoie un objet `Range` (une cellule ou une plage de cellules) et *`Texte`* est le commentaire.

Plutôt qu'une chaîne de caractères, l'argument *`Texte`* reçoit ici pour valeur l'expression `RenvoyerCommentaire(LaCellule)`. Cette expression appelle la procédure `Property Get` du même nom, en lui passant l'argument `LaCellule`. Cette procédure s'exécute et renvoie une chaîne de caractères. Celle-ci est passée à la procédure

appelante qui l'affecte, en tant que commentaire de la cellule (voir figure 5-5).



	A	B	C	D	E	F
1		Livres	Vidéo	Hi-Fi	Autres	
2	Janvier	58 963,00	45 225,00	85 485,00	45 225,00	
3	Février	45 895,00	32 568,00	79 658,00	32 568,00	
4	Mars	69 785,00	46 895,00	25 689,00	46 895,00	
5	Avril	45 214,00	54 897,00	49 652,00		
6	Mai	45 258,00	32 568,00	36 550,00		
7	Juin	38 652,00	97 632,00	56 320,00		
8	Juillet	32 510,00	45 863,00	45 520,00		
9	Août	28 952,00	32 568,00	47 965,00	32 568,00	
10	Septembre	45 693,00	94 625,00	29 865,00	94 625,00	
11	Octobre	48 956,00	31 582,00	16 495,00	31 582,00	
12	Novembre	65 920,00	21 458,00	75 632,00	21 458,00	
13	Décembre	95 120,00	12 589,00	12 589,00	12 589,00	

Figure 5-5 – *Chacune des cellules comprises dans la sélection au moment de l'exécution de la macro s'est vue affecter un commentaire.*

Procédures Property Let

Une procédure `Property Let` est structurée de la façon suivante :

```
Property Let NomDeLaProcédure (VarStockage)
   Instructions
End Property
```

Les instructions `Property Let` et `End Property` déterminent le début et la fin de la procédure. *`NomDeLaProcédure`* doit respecter les règles de Visual Basic ; ce nom est utilisé pour invoquer la procédure à partir d'une autre.

Les *`Instructions`* définissent ce qu'exécute la procédure.

`VarStockage` est la variable recevant la valeur passée par la procédure appelante. Comme vous le verrez dans l'exemple suivant, elle peut ensuite être traitée par la procédure afin de déterminer la valeur à affecter à la propriété.

La procédure suivante, `Property Let`, est en quelque sorte la procédure miroir de la `Property Get` de la section précédente. Elle affecte une couleur de remplissage à chacune des cellules de la sélection en cours : la chaîne représentant le commentaire détermine la valeur de la variable numérique qui sera affectée à la propriété `ColorIndex` de l'objet `Range` (la cellule).

```
1: Property Let CouleurDeRemplissage(LaCellule As Range)
2:   Dim IndexCouleur As Integer
3:     Select Case LaCellule.Comment.Text
4:       Case "Très mauvais"
5:         IndexCouleur = 3 'Index de la couleur Rouge
6:       Case "Mauvais"
```

```
 7:         IndexCouleur = 6 'Index de la couleur Jaune
 8:       Case "Correct"
 9:         IndexCouleur = 5'Index de la couleur Bleu
10:       Case Else
11:         IndexCouleur = xlColorIndexNone
12:     End Select
13:   LaCellule.Interior.ColorIndex = IndexCouleur
14: End Property
```

À la ligne 1, l'instruction `Property Let` déclare la procédure `CouleurDeRemplissage`, spécifiant que l'argument `LaCellule` de type `Range` doit être passé par la procédure appelante. La variable `IndexCouleur` est déclarée à l'aide de l'instruction `Dim` à la ligne 2. Une instruction de contrôle `Select Case` est ensuite utilisée pour tester la valeur de l'expression `LaCellule.Comment.Text` ; cette instruction renvoie le texte de commentaire de la cellule (lignes 3 à 12). Si ce dernier est égal à "Très mauvais", "Mauvais" ou "Correct", la variable `IndexCouleur` reçoit une valeur. Si le commentaire de la cellule ne correspond à aucun de ces cas, `IndexCouleur` se voit affecter la constante `xlColorIndexNone`. Enfin, l'instruction de la ligne 13 affecte la valeur de `IndexCouleur` à la propriété `ColorIndex` de l'objet `Interior` de la cellule, c'est-à-dire applique une couleur de remplissage à la cellule.

Pour appeler une procédure `Property Let`, utilisez une instruction assimilable à l'affectation d'une propriété (*`Propriété`* = *`Valeur`*). Le nom de la procédure placé à gauche de l'instruction d'affectation correspondra à une expression renvoyant une propriété ; l'argument requis par la procédure `Property Let`, placé à droite, déterminera la valeur affectée à la propriété.

Considérez l'exemple suivant :

```
Sub DefinirRemplissage()
  Dim LaCellule As Range
  For Each LaCellule In Selection
    CouleurDeRemplissage = LaCellule
  Next LaCellule
End Sub


Property Let CouleurDeRemplissage(LaCellule As Range)
  Dim IndexCouleur As Integer
    Select Case LaCellule.Comment.Text
      Case "Très mauvais"
        IndexCouleur = 3 'Index de la couleur Rouge
      Case "Mauvais"
        IndexCouleur = 6 'Index de la couleur Jaune
      Case "Correct"
        IndexCouleur = 5 'Index de la couleur Bleu
      Case Else
        IndexCouleur = xlColorIndexNone
    End Select
  LaCellule.Interior.ColorIndex = IndexCouleur
End Property
```

Dans la sous-routine `DefinirRemplissage`, l'instruction `CouleurDeRemplissage = LaCellule` est assimilée à une affectation de propriété. `CouleurDeRemplissage` correspond à une

expression renvoyant une propriété, et `LaCellule` détermine la valeur à affecter. La procédure `CouleurDeRemplissage` est donc appelée par cette instruction. Elle s'exécute et examine la valeur de la propriété `Text` de l'objet `Comment` de la variable `LaCellule`. La variable `IndexCouleur` reçoit une valeur, fonction du résultat retourné. Cette valeur est ensuite affectée à la propriété `ColorIndex` de l'objet `Interior` de la cellule. La procédure appelante reprend ensuite la main et procède de la même façon pour la cellule suivante de la sélection.

Syntaxe avancée

Au même titre que les instructions `Sub` et `Function`, les instructions de déclaration `Property Get` et `Property Let` peuvent être précédées des mots-clés `Private` ou `Public` et/ou `Static`, afin de spécifier si la procédure est privée ou publique et si ses variables locales sont statiques. Pour un rappel de ces concepts, reportez-vous à la section « Procédures Sub » de ce chapitre.

Les instructions de déclaration `Property Get` et `Property Let` peuvent aussi spécifier des arguments placés entre les parenthèses qui suivent le nom de la procédure. Chacun des arguments répond à la syntaxe suivante :

```
[Optional] [ByVal|ByRef] [ParamArray] NomVariable [As type] [=ValeurParDéfaut]
```

Cette syntaxe est la même que pour les déclarations `Sub` et `Function` ; reportez-vous à la section « Procédures Sub », plus haut dans ce chapitre.

Enfin, une instruction `Property Get` peut se terminer par l'argument `As Type` : c'est un moyen de préciser le type de valeur renvoyé par la procédure.

Des projets bien structurés

Alors que les feuilles sont automatiquement stockées dans le dossier UserForm de votre projet, il vous incombe de déterminer l'organisation des modules de code et des modules de classe. Vous pouvez ainsi structurer les procédures constituant votre projet, de façon à y accéder facilement. Les modules sont en quelque sorte les dossiers dans lesquels vous rangez les documents constituant vos applications VBA.

Veillez à structurer le code en procédures distinctes. Si une application VBA peut être contenue dans une seule procédure, la division des tâches complexes en plusieurs procédures distinctes qui s'appelleront est fortement recommandée. Le code ainsi segmenté sera plus facile à gérer et le débogage considérablement simplifié. En outre, organisez les procédures d'un projet

dans des modules cohérents, en réunissant celles qui ont des aspects communs.

Ajouter un module

Vous serez probablement amené à développer des applications VBA distinctes au sein d'un même projet. Dans ce cas-là, il est important de regrouper leurs procédures dans des modules séparés. Si vous les stockez toutes au sein d'un même module, sans aucune distinction, vous risquez d'être rapidement dépassé par un nombre important de procédures dont vous serez incapable de définir les rapports.

Pour créer un module standard ou un module de classe, procédez comme suit :

1. Lancez Visual Basic Editor à partir d'Excel (Alt+F11) – le document hôte du projet doit être ouvert.
2. Affichez l'Explorateur de projet (Ctrl+R). Si plusieurs projets sont accessibles, cliquez sur n'importe quel élément du projet auquel vous souhaitez ajouter un module, afin de l'activer.
3. Pour ajouter un module au projet actif et ouvrir sa fenêtre Code, choisissez l'une des trois méthodes suivantes :
   - Cliquez-droit et, dans le menu contextuel qui s'affiche, sélectionnez Insertion. Dans le sous-menu, sélectionnez Module ou Module de classe.
   - Ouvrez le menu Insertion et choisissez la commande Module ou Module de classe.
   - Cliquez sur la flèche du bouton Ajouter… de la barre d'outils Standard. Dans le menu qui s'affiche, sélectionnez Module ou Module de classe.

   Le module inséré est automatiquement nommé : Module1 (ou Module2 si Module1 existe déjà…) pour un module standard et Class1 (Class2 si Class1 existe déjà…) pour un module de classe.
4. Ouvrez la fenêtre Propriétés (F4) du nouveau module et donnez-lui un nom représentatif. Ce dernier, qui apparaît dans la barre de titre de la fenêtre Code et dans l'Explorateur de projet, est automatiquement mis à jour (voir figure 5-6).

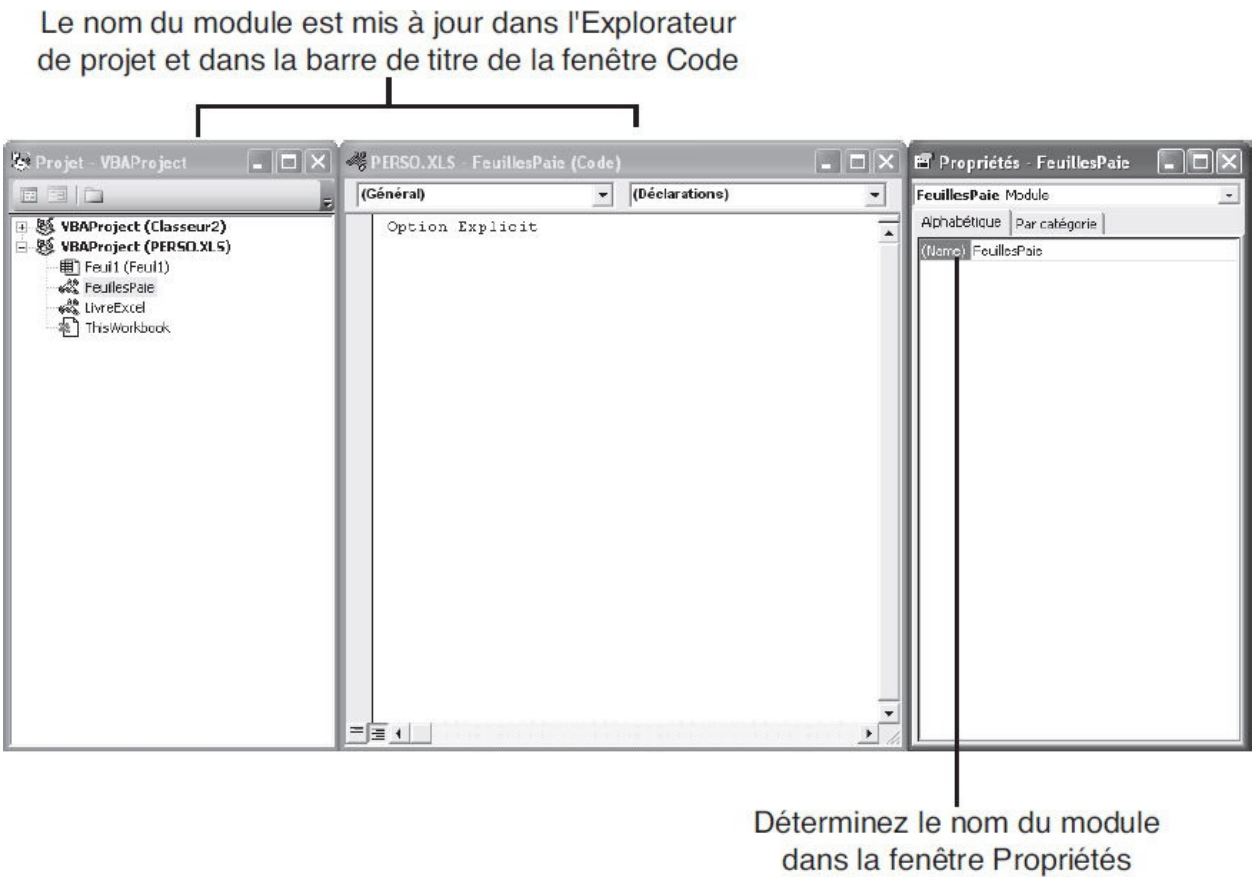


Figure 5-6 – *Choisissez des noms représentatifs pour vos modules.*

Supprimer un module

Pour supprimer un module ou une feuille d'un projet, procédez comme suit :

1. Sélectionnez ce que vous voulez supprimer dans l'Explorateur de projet.
2. Cliquez-droit et, dans le menu contextuel qui s'affiche, sélectionnez Supprimer *Module*.

   Visual Basic Editor affiche une boîte de dialogue vous proposant d'exporter le module avant la suppression (voir figure 5-7).

Figure 5-7 – *Pour sauvegarder les informations contenues dans un module avant sa suppression, exportez-le.*

3. Si vous souhaitez supprimer définitivement le module, choisissez Non.
4. Pour sauvegarder le module afin de pouvoir le récupérer en cas de nécessité, choisissez Oui. La boîte de dialogue Exporter un fichier s'affiche (voir figure 5-8). Le type du fichier varie selon l'élément supprimé :
   - Les modules standards, ou modules de code, sont exportés sous forme de fichiers Basic portant l'extension .bas.
   - Les modules de classe sont exportés sous forme de fichiers Classe portant l'extension .cls.
   - Les feuilles sont exportées sous la forme de fichiers Feuille portant l'extension .frm.
6. Indiquez le répertoire et le nom d'enregistrement, puis cliquez sur le bouton Enregistrer.

Figure 5-8 – *Indiquez le nom du fichier exporté et son dossier d'enregistrement.*

Info

Pour sauvegarder un module sans le supprimer, sélectionnez la commande Fichier > Exporter un fichier. Vous pourrez ensuite l'importer dans n'importe quel projet en choisissant Fichier > Importer un fichier.

Créer une procédure

Pour créer une procédure, activez la fenêtre Code du module dans lequel elle sera stockée. Vous pouvez ensuite écrire directement l'instruction de déclaration ou utiliser la boîte de dialogue Ajouter une procédure.

Écrire l'instruction de déclaration

1. Dans la fenêtre Code du module voulu, placez le point d'insertion à l'endroit où vous souhaitez ajouter une procédure.

> **Attention**
> Les procédures ne peuvent être imbriquées. Veillez donc à ce que le point d'insertion se trouve à l'extérieur de toute procédure.

2. Saisissez les éventuels mots-clés précisant la portée (`Public` ou `Private`) et le comportement des variables locales (`Static`).
3. Tapez l'instruction de déclaration correspondant au type de procédure que vous souhaitez créer, soit `Sub`, `Function`, `Property Get`, `Property Let` ou `Property Set`.
4. Saisissez le nom de la procédure.
5. Indiquez ensuite les éventuels arguments entre parenthèses.
6. Tapez sur la touche Entrée afin de placer un retour chariot.

   Visual Basic Editor ajoute automatiquement l'instruction `End` correspondante (`End Sub`, `End Function` ou `End Property`). Si vous n'avez indiqué aucun argument, une parenthèse ouvrante, immédiatement suivie d'une parenthèse fermante, est automatiquement ajoutée derrière le nom de la procédure (voir figure 5-9).
7. Saisissez le code de votre procédure entre ces instructions d'encadrement.

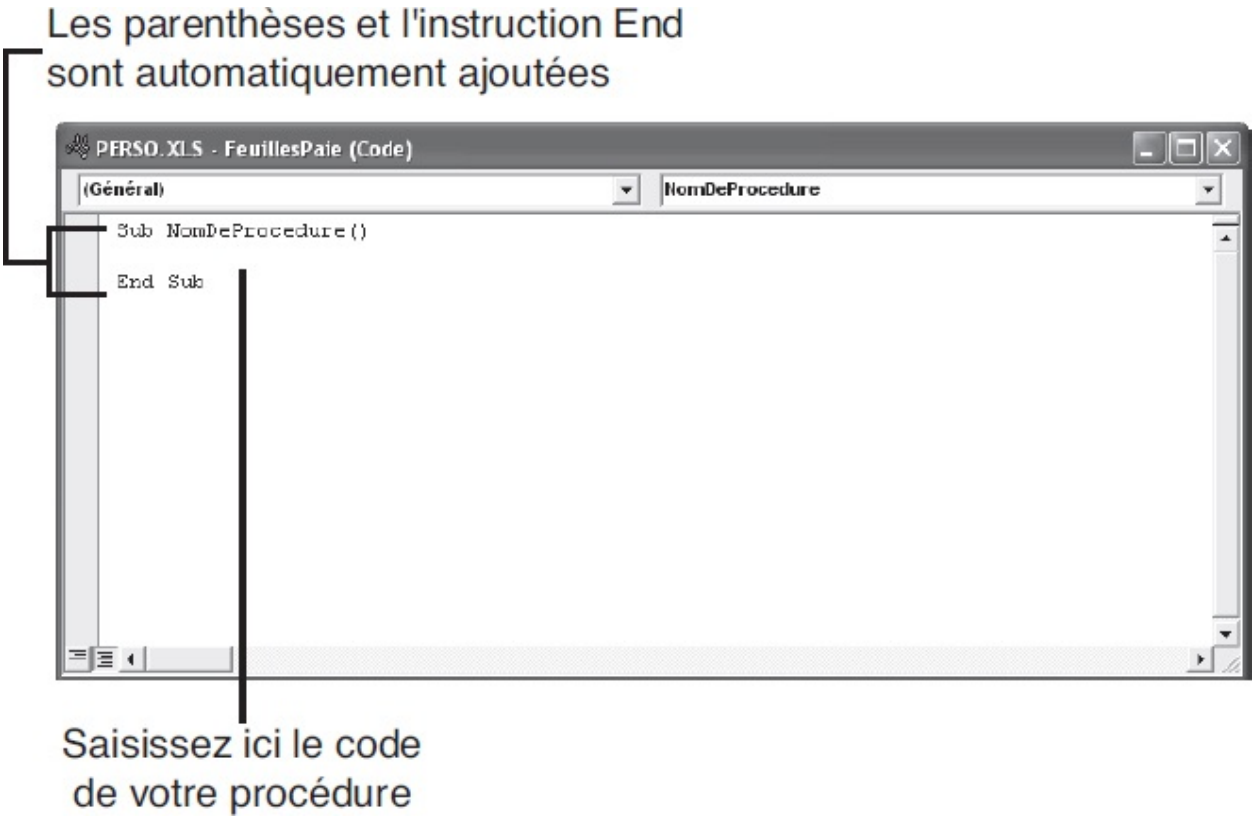


Figure 5-9 – *Saisissez l'instruction de déclaration de la procédure, et Visual Basic Editor insérera l'instruction End correspondante.*

Vous pouvez aussi préciser le comportement et la disponibilité de la procédure en ajoutant, par exemple, le mot-clé Static devant l'instruction de déclaration. Pour plus de précisions, reportez-vous au chapitres 6.

La boîte de dialogue Ajouter une procédure

1. Activez la fenêtre Code du module dans lequel vous souhaitez insérer une procédure.

2. Cliquez sur la flèche située à droite du bouton Ajouter… de la barre d'outils Standard et, dans le menu qui s'affiche, sélectionnez la commande Procédure.

   La boîte de dialogue présentée à la figure 5-10 s'affiche.

Info

La commande Procédure n'est pas disponible si la fenêtre active n'est pas une fenêtre Code.

Figure 5-10 – *La boîte de dialogue Ajouter une procédure.*

3. Entrez les informations décrivant la procédure : son nom, son type (bouton radio), sa portée ; et cochez éventuellement la case Toutes les variables locales statiques.
4. Cliquez sur le bouton OK. La boîte de dialogue se ferme et les instructions d'encadrement de la procédure sont insérées dans la partie inférieure de la fenêtre Code.
5. Saisissez les éventuels arguments entre les parenthèses.
6. Saisissez le code de votre procédure entre l'instruction de déclaration et l'instruction `End` correspondante.

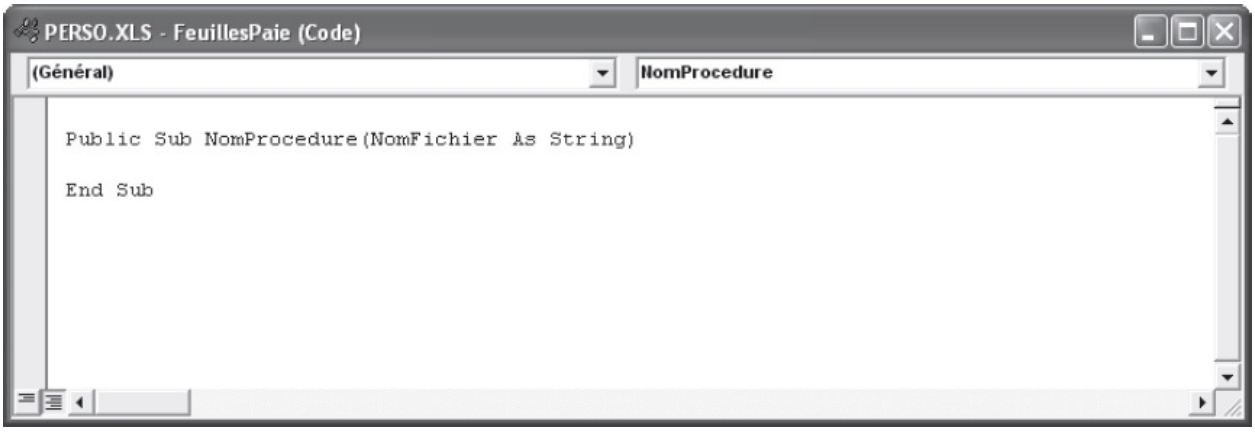


Figure 5-11 – *Les instructions d'encadrement de la procédure définie dans la*

boîte de dialogue Ajouter une procédure sont automatiquement insérées dans la partie inférieure de la fenêtre Code.

> **Info**
> Lorsque vous sélectionnez le type `Property`, les instructions d'encadrement d'une procédure `Property Get` et d'une procédure `Property Let` sont insérées.

La notion de portée

La portée d'une procédure est essentielle, puisqu'elle en détermine l'accessibilité dans le reste du projet. *Publique*, elle est visible pour n'importe quelle procédure du projet, quel que soit son module de stockage ; *privée*, elle est invisible pour les procédures autres que celles de son module.

La portée d'une procédure est déterminée dans son instruction de déclaration, grâce au mot-clé `Public` ou `Private`.

Par défaut, les procédures des modules de code et de classe sont publiques. Il n'est donc pas utile de placer le mot-clé `Public` dans ce cas. En revanche, si vous souhaitez déclarer une procédure privée, vous devez ajouter le mot-clé `Private` devant l'instruction de déclaration.

Une procédure événementielle est, par définition, privée. Elle ne peut en effet pas être appelée par une autre procédure, puisque seul l'événement spécifié est capable de la déclencher. Comme vous le verrez au chapitres 14, lorsque vous créez des procédures événementielles, le mot-clé `Private` est automatiquement inséré.

Les deux instructions suivantes déclarent une procédure `Sub` de portée publique :

```
Sub MaProcédurePublique()
  Instructions
End Sub


Public Sub MaProcédurePublique()
  Instructions
End Sub
```

L'instruction suivante déclare une procédure `Sub` de portée privée :

```
Private Sub MaProcédurePrivée()
  Instructions
End Sub
```

Écriture et mise en forme du code

Les instructions constituant une procédure sont autonomes et sont généralement écrites une par ligne. Placez le curseur sur une ligne vierge et saisissez le texte au clavier. Lorsque vous avez fini, tapez sur la touche Entrée pour passer à l'instruction suivante.

Caractère de continuité de ligne

Une instruction peut cependant nécessiter plusieurs lignes de code : placez le caractère de continuité de ligne – le trait de soulignement (_) – afin d'indiquer que l'instruction se poursuit sur la ligne suivante. Cela facilite dans certains cas la lecture du code, en évitant le recours à la barre de défilement horizontale.

Ainsi, la procédure suivante est composée des instructions de déclaration `Sub` et `End Sub` et d'une unique instruction exécutable `MsgBox` répartie sur plusieurs lignes :

```
Sub MaProcédure()
  MsgBox ("Cette instruction affiche une boîte de dialogue " _
  & "dans laquelle est affiché ce très long message, que nous avons " _
  & "réparti sur quatre lignes à l'aide du caractère de continuité " _
  & "de ligne")
End Sub
```

Notez que le message affiché par la fonction `MsgBox` a été séparé en plusieurs chaînes de caractères concaténées à l'aide de l'opérateur `&`. Le caractère de continuité de ligne ne peut en effet être utilisé qu'entre des éléments distincts d'une instruction. Autrement dit, une même chaîne de caractères, une expression, un mot-clé ou encore une constante ne peuvent être écrits sur plusieurs lignes.

Les commentaires

Un *commentaire* est une indication destinée à faciliter la lecture du code en décrivant les tâches qu'exécute une instruction, la date de création d'une procédure, etc.

L'insertion de commentaires dans le texte d'une procédure est utile, particulièrement si elle comprend un grand nombre d'instructions dont vous souhaitez reconnaître rapidement les fonctions respectives.

Si vous souhaitez insérer des commentaires dans le texte d'une procédure, il faut bien évidemment que ceux-ci soient reconnus en tant que tels et non en tant qu'instructions, ce qui générerait une erreur lors de l'exécution de la

procédure.

Visual Basic considère le texte en tant que commentaire et l'ignore lors de l'exécution d'une procédure s'il est précédé :

- d'une apostrophe (') ;
- du mot-clé `REM`.

Par défaut, les commentaires apparaissent en vert dans Visual Basic Editor. Pour en modifier la couleur d'affichage, reportez-vous à la section « Un code tout en couleurs », plus loin dans ce chapitre.

Utiliser l'apostrophe

L'utilisation de l'apostrophe pour marquer les commentaires permet de les placer à n'importe quel endroit du texte. Vous pouvez ainsi insérer un commentaire sur la même ligne que l'instruction concernée en plaçant autant d'espaces que vous le souhaitez entre celle-ci et l'apostrophe. Cette façon de faire est particulièrement intéressante et efficace pour les instructions dont la syntaxe est courte, puisqu'il est possible d'aligner les différents commentaires (figure 5-12).

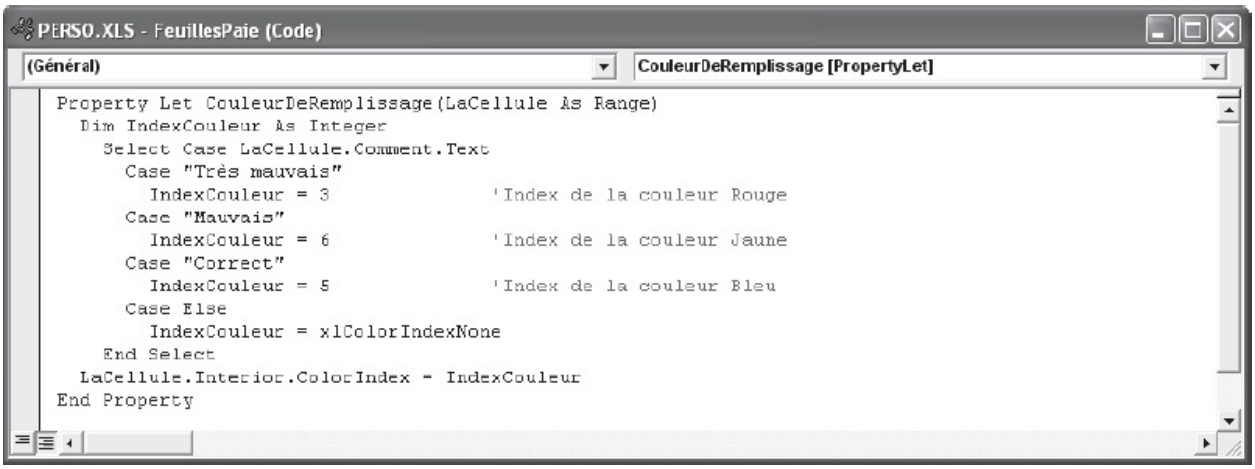


Figure 5-12 – *L'apostrophe permet d'aligner les commentaires.*

Utiliser *REM*

La syntaxe `REM` joue le même rôle que l'apostrophe. Cependant, contrairement à celle-ci, le marqueur `REM` ne peut être accolé à l'instruction qu'il commente mais doit toujours être placé en début de ligne. Il sera donc utilisé de préférence pour commenter des blocs d'instructions (figure 5-13).

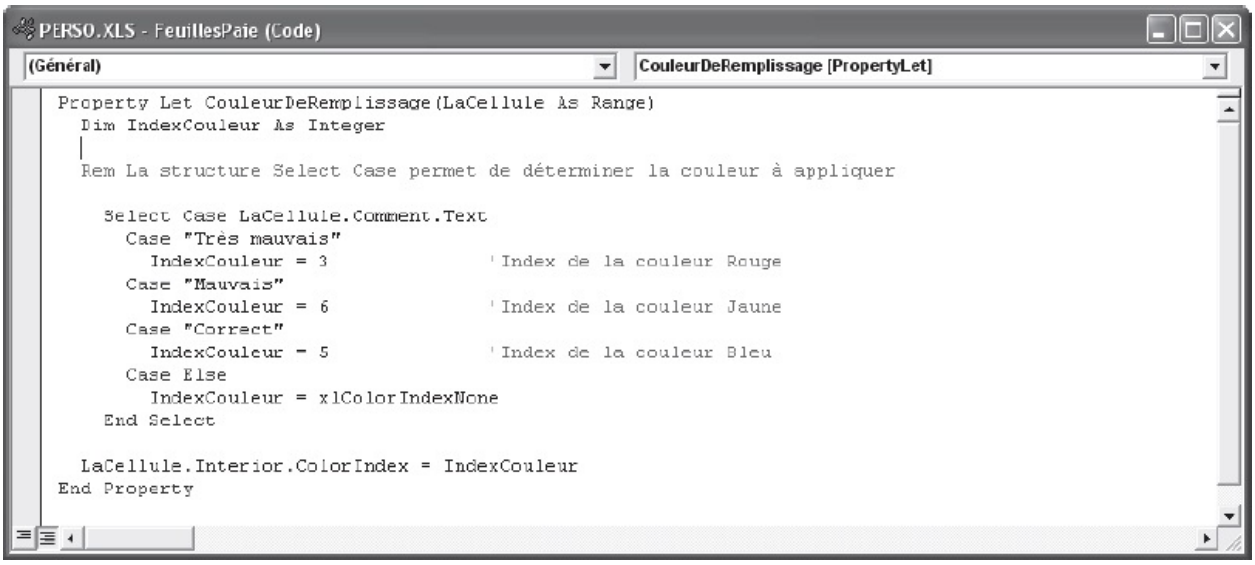

PERSO.XLS - FeuillesPaie (Code)
(Général)
CouleurDeRemplissage [PropertyLet]
Property Let CouleurDeRemplissage(LaCellule As Range)
Dim IndexCouleur As Integer
Rem La structure Select Case permet de déterminer la couleur à appliquer
Select Case LaCellule.Comment.Text
Case "Très mauvais"
IndexCouleur = 3 'Index de la couleur Rouge
Case "Mauvais"
IndexCouleur = 6 'Index de la couleur Jaune
Case "Correct"
IndexCouleur = 5 'Index de la couleur Bleu
Case Else
IndexCouleur = xlColorIndexNone
End Select
LaCellule.Interior.ColorIndex = IndexCouleur
End Property

Figure 5-13 – *L'instruction REM ne peut être placée qu'en début de ligne.*

Conseil

Utilisez des caractères facilement discernables pour faire ressortir les commentaires :

```
'*******************************
'****Ceci est un commentaire****
'*******************************
```

Commenter un bloc d'instructions

Lorsque vous testerez le comportement des applications VBA, il se révélera parfois nécessaire de placer en commentaire un bloc d'instructions que vous ne souhaitez pas voir s'exécuter lors de cette phase. Cliquez pour cela sur le bouton Commenter bloc de la barre d'outils Édition, qui place une apostrophe devant chaque ligne de la sélection dans la fenêtre Code active.

Pour réactiver un bloc d'instructions commentées, sélectionnez-le et cliquez sur le bouton Ne pas commenter de la barre d'outils Édition.

Info

Lorsque vous enregistrez une macro dans l'application hôte, des lignes de commentaires indiquant la date d'enregistrement sont placées derrière l'instruction de déclaration de la procédure.

Mise en forme du code

La mise en forme du code consiste à appliquer des retraits de ligne variables aux instructions et à laisser des espaces entre les blocs pour améliorer la lisibilité du code et en faciliter l'interprétation. On applique en général un même retrait aux instructions s'exécutant à un même niveau dans la procédure ou appartenant à une même structure. Lorsque des instructions sont imbriquées dans d'autres, on leur applique un retrait supplémentaire par rapport à ces dernières. La hiérarchie qui régit les instructions d'une procédure est ainsi clairement visible, ce qui en simplifie la lecture (voir figure 5-14).

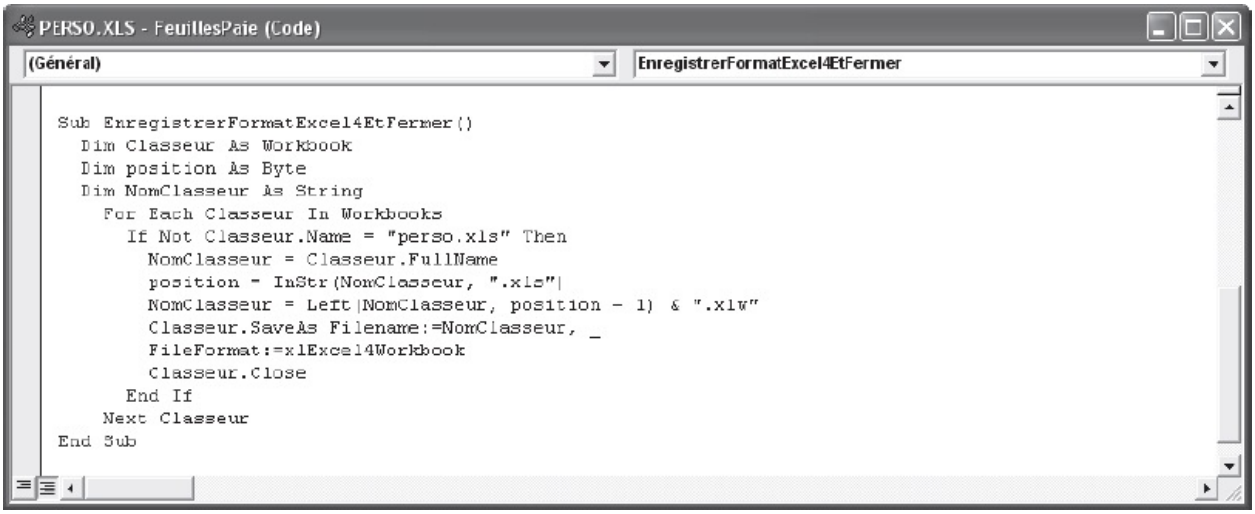


Figure 5-14 – *Utilisez les retraits de ligne pour améliorer la lisibilité de votre code.*

Définition

Des instructions sont dites imbriquées lorsque leur exécution s'effectue à l'intérieur d'un autre bloc d'instructions. Vous pouvez, par exemple, créer une instruction conditionnelle qui vérifie qu'une plage de cellules contient des valeurs numériques et y imbriquer une autre structure conditionnelle qui effectuera des tâches déterminées. Les structures de contrôle et leur imbrication sont étudiées au chapitres 7.

Les retraits de ligne sont indifféremment composés d'espaces ou de tabulations. L'éditeur peut aussi être paramétré de façon que, lorsqu'un retrait est ajouté à une ligne, il soit appliqué aussi à la ligne suivante :

1. Choisissez la commande Options du menu Outils et sélectionnez l'onglet Éditeur.
2. Cochez la case Retrait automatique, puis définissez une valeur comprise entre 1 et 32 dans la zone Retrait de la tabulation. Cela correspond au

nombre d'espaces qui seront appliqués lors de la frappe de la touche Tabulation.

3. Cliquez sur OK pour valider les paramètres définis.

Pour augmenter ou diminuer simultanément le retrait de ligne de plusieurs instructions, sélectionnez celles-ci, puis cliquez sur le bouton Retrait ou sur le bouton Retrait négatif de la barre d'outils Édition.

Un code tout en couleurs

Toujours dans l'optique de faciliter l'interprétation du code, les éléments constitutifs d'une procédure sont affichés dans différentes couleurs, chacune identifiant une catégorie spécifique du langage. Par exemple, les commentaires sont affichés par défaut en vert, tandis que les mots-clés du langage apparaissent en bleu et les erreurs de syntaxe en rouge (voir figure 5-15).

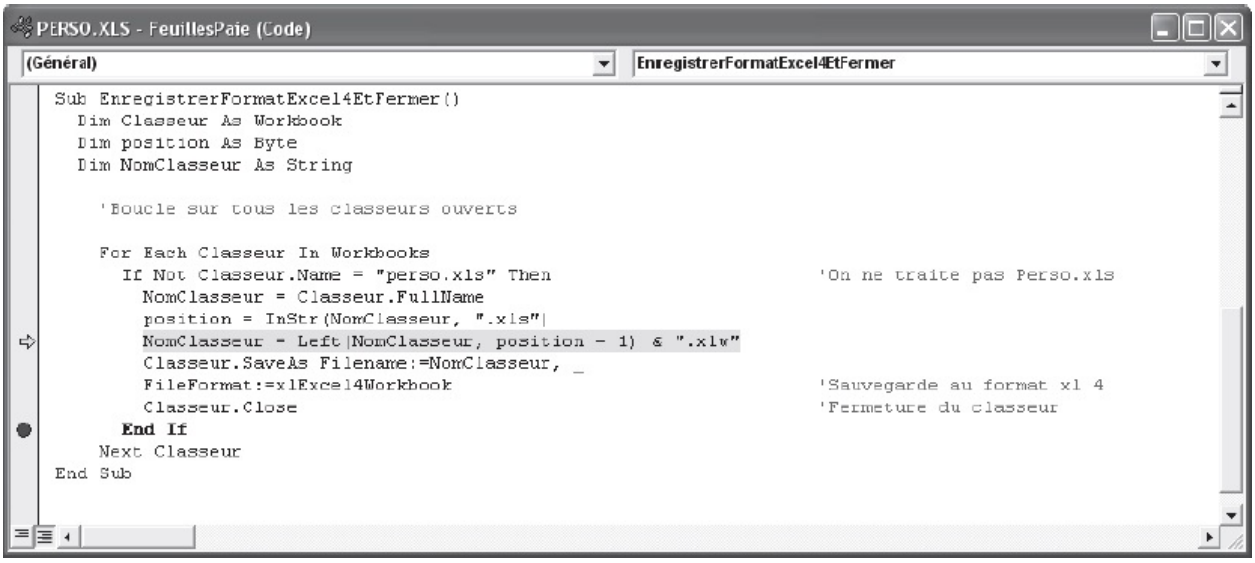

PERSO.XLS - FeuillesPaie (Code)
(Général)
EnregistrerFormatExcel4EtFermer
Sub EnregistrerFormatExcel4EtFermer()
Dim Classeur As Workbook
Dim position As Byte
Dim NomClasseur As String
'Boucle sur tous les classeurs ouverts
For Each Classeur In Workbooks
If Not Classeur.Name = "perso.xls" Then 'On ne traite pas Perso.xls
NomClasseur = Classeur.FullName
position = InStr(NomClasseur, ".xls")
NomClasseur = Left(NomClasseur, position - 1) & ".xlw"
Classeur.SaveAs Filename:=NomClasseur, _
FileFormat:=xlExcel4Workbook 'Sauvegarde au format xl 4
Classeur.Close 'Fermeture du classeur
End If
Next Classeur
End Sub

Figure 5-15 – *Les couleurs, retranscrites ici en niveaux de gris, déterminent les catégories de texte et mettent en valeur les erreurs de syntaxe.*

Pour définir vos propres paramètres d'affichage du texte dans la fenêtre Code, procédez comme suit :

1. Choisissez la commande Options du menu Outils, puis activez l'onglet Format de l'éditeur (voir figure 5-16).
2. Dans la zone Couleur de code, sélectionnez la catégorie dont vous souhaitez modifier l'affichage. Déterminez ensuite les options de votre choix dans les listes déroulantes suivantes :

- Premier plan. Définit la couleur de premier plan. Lorsque le texte n'est pas sélectionné, il s'agit de la couleur des caractères.
- Arrière-plan. Détermine la couleur d'arrière-plan.
- Indicateurs. Définit la couleur des indicateurs apparaissant en marge, pour les catégories du langage affichant ce type d'indicateur.

3. Modifiez éventuellement la police d'affichage du code dans la liste déroulante Police et dans la zone Taille.
4. Pour masquer la barre des indicateurs en marge, décochez la case correspondante. Vous gagnerez de l'espace pour afficher le code.
5. Cliquez sur OK pour valider les paramètres d'affichage définis.

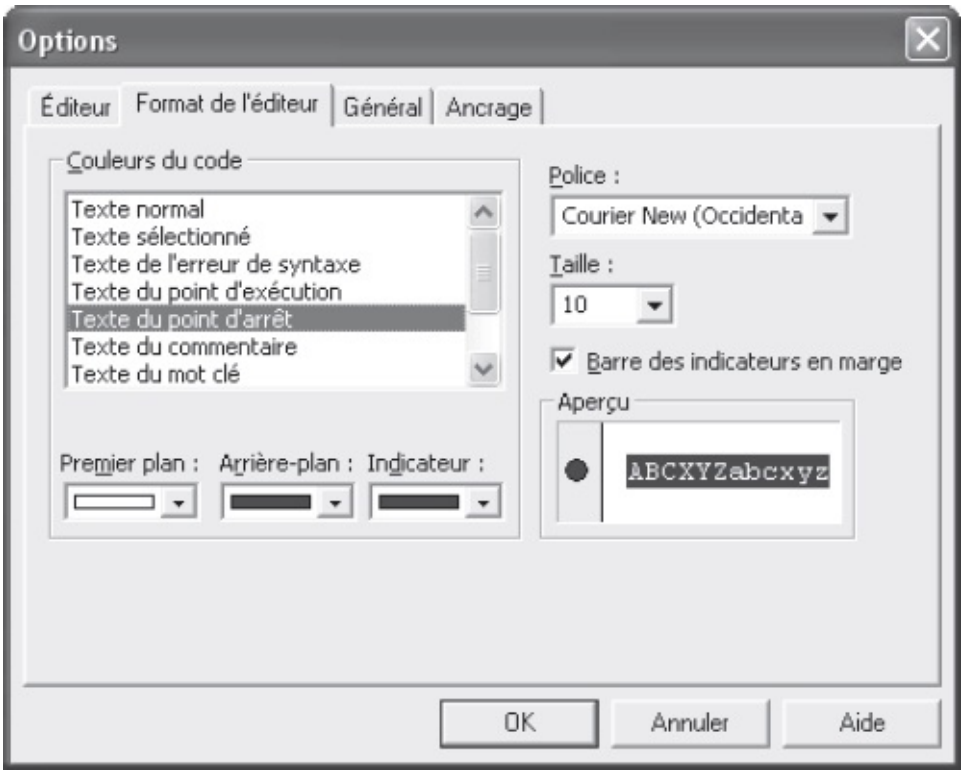


Figure 5-16 – *L'onglet Format de l'éditeur permet de personnaliser l'affichage du code.*

Conseil

Les paramètres d'affichage du code tels qu'ils sont définis par défaut dans Visual Basic Editor assurent une lecture confortable à l'écran. Si vous souhaitez cependant les personnaliser, veillez à conserver cette qualité de lecture.

Déplacer une procédure

Il vous arrivera de souhaiter mettre de l'ordre dans vos projets VBA. Vous serez alors probablement amené à déplacer des procédures d'un module à un autre, afin de structurer le tout de façon cohérente. Utilisez la technique du glisser-déplacer :

1. Ouvrez les fenêtres Code concernées et affichez-les simultanément à l'aide de la commande Mosaïque horizontale ou Mosaïque verticale du menu Fenêtre.
2. Sélectionnez la procédure à déplacer.
3. Cliquez sur le texte ainsi sélectionné et, tout en maintenant le bouton de la souris enfoncé, déplacez le curseur vers la fenêtre Code voulue. Une barre verticale grise indique où la procédure sera déposée.
4. Relâchez le bouton de la souris.

Info

S'il est impossible d'effectuer un glisser-déplacer dans la fenêtre Code, sélectionnez Outils puis Options et cochez la case Glisser-déplacer pour l'édition de texte de l'onglet Éditeur.

Conseil

N'oubliez pas que l'Enregistreur de macro peut réduire de façon considérable le travail d'écriture de code. N'hésitez pas à enregistrer les tâches qui peuvent l'être ; ajoutez ensuite les autres instructions dans la fenêtre Code de la macro.

Si vous devez ajouter des instructions à une macro, vous pouvez enregistrer les commandes correspondantes dans une nouvelle macro, puis coller le code ainsi généré à l'emplacement voulu.

Appel et sortie d'une procédure

La division d'un programme en plusieurs modules effectuant des tâches précises en améliore la lisibilité et facilite les éventuels débogages. Pour exécuter le programme, l'utilisateur lance une procédure. Celle-ci effectue alors les tâches pour lesquelles elle a été définie et appelle éventuellement d'autres procédures qui prendront en charge l'exécution d'autres tâches. Ces procédures pourront elles-mêmes en appeler d'autres, et ainsi de suite. Lorsqu'une procédure en appelle une autre, cette dernière s'exécute, puis la procédure appelante reprend la main et se poursuit avec l'instruction qui suit l'instruction d'appel.

Appel d’une procédure Sub

Appeler une procédure consiste à lui demander de s’exécuter à partir d’une autre. La procédure appelée est exécutée, puis la procédure appelante reprend la main.

Pour appeler une procédure de type `Sub`, utilisez le mot-clé `Call`, selon la syntaxe suivante :

```
Call NomProcédure
```

Conseil

Une instruction contenant simplement le nom d’une procédure suffit à appeler cette dernière. L’omission du mot-clé `Call` est cependant déconseillée, car le code y perd en lisibilité.

Considérez l’exemple suivant :

```
Sub AppelsDeProcédures()
  MsgBox "1er message de la procédure 1"
  Call Procédure2
  MsgBox "2e message de la procédure 1"
End Sub


Sub Procédure2()
  MsgBox "1er message de la procédure 2"
  Call Procédure3
  MsgBox "2e message de la procédure 2"
End Sub


Sub Procédure3()
  MsgBox "1er message de la procédure 3"
  MsgBox "2e message de la procédure 3"
End Sub
```

L’exécution de la procédure `AppelsDeProcédures` entraîne l’affichage successif des messages suivants :

- 1er message de la procédure 1
- 1er message de la procédure 2
- 1er message de la procédure 3
- 2e message de la procédure 3
- 2e message de la procédure 2
- 2e message de la procédure 1

La première instruction de `AppelsDeProcédures` affiche un message. `Procédure2` est ensuite appelée à l’aide de l’instruction `Call`. Elle affiche un message, puis appelle à son tour `Procédure3`. Celle-ci s’exécute et affiche consécutivement deux

messages. Elle se termine alors et la procédure appelante, `Procédure2`, reprend la main. Elle se poursuit avec l'instruction placée immédiatement sous l'appel et affiche de nouveau un message. Elle se termine à son tour et rend la main à `AppelsDeProcédures`, qui affiche un ultime message et se termine.

> **Attention**
> Pour appeler une procédure, celle-ci doit être visible pour la procédure appelante. Pour plus d'informations, reportez-vous à la section « La notion de portée », plus haut dans ce chapitre.

Appels de procédures Function et Property

Les procédures `Function` et `Property` ont pour première fonction de renvoyer une valeur – vous pouvez y placer des instructions effectuant des tâches précises dans un document, mais il est préférable de réserver ces opérations aux procédures `Sub`. Pour appeler une procédure `Function` ou `Property`, il suffit de placer son nom dans une expression, à l'emplacement où une valeur est attendue ; elle s'exécute alors et se voit affecter une valeur qui se substitue à son appel dans la procédure appelante.

Pour des exemples d'appels de ce type de procédures, reportez-vous aux sections « Procédures Function » et « Procédures Property », plus haut dans ce chapitre.

Passage d'arguments

Lorsqu'une procédure est appelée, il est souvent nécessaire de lui passer des valeurs. Elle les exploite alors et peut à son tour passer des valeurs à une autre procédure. Les arguments admis doivent apparaître entre parenthèses dans l'instruction de déclaration de la procédure appelée. Ils sont facultatifs ou obligatoires, de type défini ou non.

L'argument passé peut être une valeur ou toute expression renvoyant une valeur d'un même type de données – voir au chapitres 6.

Pour une revue détaillée de la syntaxe de déclaration des arguments d'une procédure, reportez-vous au tableau 5-1.

Passage d'arguments par ordre d'apparition

Pour passer des arguments à une procédure, il suffit d'en lister les valeurs entre parenthèses dans l'instruction d'appel, en les séparant par des virgules et

en respectant leur ordre d'apparition dans la déclaration de la procédure appelée. Plus concrètement, pour passer des arguments à la procédure suivante :

```
Sub MaProcédure(Arg1, Arg2, Arg3)
```

utilisez une instruction `Call` de cette façon :

```
Call MaProcédure(ValArg1, ValArg2, ValArg3)
```

où *`ValArg1`*, *`ValArg2`* et *`ValArg3`* sont respectivement les valeurs que prendront les arguments `Arg1`, `Arg2` et `Arg3`.

Certains arguments sont optionnels ; la procédure appelée s'exécutera correctement sans qu'ils lui soient passés. Pour ignorer un argument, placez deux virgules consécutives dans l'instruction d'appel correspondant à l'emplacement de l'argument dans l'instruction de déclaration de la procédure appelée. Par exemple, l'instruction utilisée pour passer les arguments `Arg1` et `Arg3` à la procédure `MaProcédure` sera :

```
Call MaProcédure(ValArg1, , ValArg3)
```

Attention

N'oubliez pas de placer une virgule pour chaque argument non transmis. Un tel oubli engendrerait un décalage dans le passage de valeur à la procédure appelée. L'instruction suivante :

`Call MaProcédure(ValArg1, ValArg3)`

passe respectivement les valeurs `ValArg1` et `ValArg3` aux arguments `Arg1` et `Arg2` de `MaProcédure`.

Conseil

Pour vérifier si des arguments facultatifs ont été omis lors de l'appel de la procédure, utilisez la fonction `IsMissing` selon la syntaxe suivante :

`IsMissing(`*`NomArgument`*`)`

La valeur `True` est renvoyée si aucune valeur correspondant à l'argument n'a été passée à la procédure ; `False` dans le cas contraire.

Attention

La fonction `IsMissing` fonctionne avec des valeurs de type `Variant`. Si vous testez un argument d'un autre type, `IsMissing` renverra toujours `False`, qu'il ait été passé ou non.

Arguments nommés

Le passage d'arguments selon leur ordre d'apparition dans l'instruction de déclaration de la procédure appelée est parfois périlleux. En particulier, lorsque des arguments facultatifs sont omis, une simple virgule oubliée dans

l'instruction d'appel pouvant provoquer une erreur du programme ou, pire, des résultats erronés dont il sera difficile de détecter la source.

L'utilisation des *arguments nommés* évite de tels problèmes, puisqu'elle implique de nommer la valeur passée à la procédure afin d'éviter toute ambiguïté. Le nom des arguments nommés est reconnu par le programme. Lorsque vous déclarez une procédure, les arguments qui lui sont affectés deviennent des arguments nommés. Il est alors possible de leur affecter des valeurs, dans l'ordre de votre choix, en mentionnant le nom de l'argument dans l'instruction d'appel, selon la syntaxe suivante :

```
Call NomProcédure (ArgNommé:=valeur, AutreArgNommé:=valeur)
```

Dans l'exemple suivant, `PassageArgumentsNommés` appelle `MaProcédure` et lui transmet les arguments nommés `NomFichier` et `Propriétaire` :

```
Sub PassageArgumentsNommés()
  Call MaProcédure(Propriétaire:=ActiveWorkbook.BuiltinDocumentProperties(3), _
                   NomFichier:=ActiveWorkbook.Name)
End Sub


Sub MaProcédure(NomFichier As String, Propriétaire As String)
  MsgBox "L'auteur du classeur " & NomFichier & " est " & _
         Propriétaire
End Sub
```

`MaProcédure` accepte les arguments `NomFichier` et `Propriétaire`, tous deux de type `String` (chaîne de caractères). `PassageArgumentsNommés` lui passe ces arguments, sans tenir compte de leur ordre d'apparition, mais en utilisant leurs noms. L'argument `Propriétaire` reçoit pour valeur la chaîne renvoyée par l'expression `ActiveWorkbook.BuiltinDocumentProperties(3)`. La propriété `BuiltinDocumentProperties` renvoie une collection `DocumentProperties` qui représente toutes les propriétés de document prédéfinies. L'index 3 permet de ne renvoyer que le troisième membre de la collection, en l'occurrence le nom de l'auteur. `NomFichier` se voit affecter la chaîne retournée par l'expression `ActiveWorkbook.Name` qui renvoie le nom du classeur actif. Ces arguments sont exploités par la procédure appelée pour afficher la boîte de dialogue représentée à la figure 5-17.

Figure 5-17 – *Utilisez les noms des arguments pour passer des informations à*

une procédure dans un ordre aléatoire.

Info

Les arguments nommés ne sont pas une spécificité des procédures. Pour la plupart, les fonctions intégrées de Visual Basic les intègrent aussi.

Sortie d'une procédure

Il peut être utile de quitter une procédure avant la fin de son exécution. Pour ce faire, utilisez le mot-clé `Exit` suivi du mot-clé déterminant le type de la procédure : `Exit Sub`, `Exit Property` ou `Exit Function`. Le programme se poursuit avec l'instruction suivant immédiatement celle qui a appelé la procédure quittée.

Dans l'exemple suivant, une boîte de dialogue s'affiche à l'aide de la fonction `MsgBox` et demande à l'utilisateur s'il souhaite exécuter de nouveau le programme.

```
Sub MonProgramme()
  Instructions
  Call AutreProcédure
  Instructions
  Recommencer = MsgBox ("Recommencer l'opération ?", vbYesNo + vbQuestion)
    If Recommencer=vbYes Then
      Call MonProgramme
    Else
      Exit Sub
    End If
End Sub
```

`MonProgramme` exécute des instructions, puis appelle une autre procédure. Elle reprend ensuite la main et exécute une autre série d'instructions. Une boîte de dialogue s'affiche, proposant à l'utilisateur de réitérer l'opération. Une structure de contrôle `If…End If` est utilisée pour déterminer le comportement du programme, en fonction de la réponse de l'utilisateur. S'il choisit le bouton Oui, la procédure s'appelle elle-même et s'exécute de nouveau ; s'il choisit Non, une instruction `Exit` entraîne la sortie de la procédure et le programme se termine.

Info

Les structures de contrôle et les fonctions `MsgBox` et `InputBox` sont étudiées au chapitres 7.

Sortie d'un programme

Deux instructions interrompent l'exécution d'un programme : `End` et `Stop`.

`End` met fin à l'exécution d'un programme et libère l'ensemble des ressources mémoire qu'il utilise. Les variables perdent leur valeur, les feuilles UserForm sont déchargées et les éventuels classeurs ouverts par le programme à l'aide de la méthode `Open` sont fermés.

Si votre programme utilise des variables objets ou charge des feuilles UserForm, `End` réinitialise l'ensemble des valeurs, vous préservant ainsi d'éventuels problèmes de mémoire et vous ramenant aux mêmes conditions initiales à chaque exécution. À l'inverse, quand vous ne mettez pas fin au programme par un `End` et ne libérez pas les ressources qu'il utilisait, les valeurs telles que les données entrées dans une feuille UserForm restent chargées en mémoire ; si la feuille est à nouveau affichée au cours de la même session Excel, elle le sera avec ces valeurs.

L'instruction `Stop` est utilisée dans le cadre du débogage (voir chapitres 10) et de l'analyse des programmes VBA : elle place l'exécution en mode Arrêt. Lorsqu'un programme atteint un `Stop`, il s'interrompt et Visual Basic Editor s'ouvre sur la fenêtre Code du module contenant l'instruction `Stop`, qui apparaît en surbrillance. Les ressources mémoire ne sont pas libérées et les variables conservent leurs valeurs. Vous pouvez poursuivre l'exécution pas à pas, écrire des instructions dans la fenêtre Exécution, afin d'évaluer ou de tester votre programme, ou encore poursuivre l'exécution du programme. Les outils et les techniques de débogage des programmes VBA constituent le sujet du chapitres 10.

Exécuter du code

À partir de Visual Basic Editor, placez le curseur dans la procédure que vous souhaitez exécuter, puis cliquez sur le bouton Exécuter de la barre d'outils Standard, ou choisissez la commande Exécuter Sub/UserForm du menu Exécution, ou appuyez sur la touche F5.

Seule une procédure `Sub` ne nécessitant pas d'arguments est accessible par la boîte de dialogue Macro d'Excel. Pour l'exécuter à partir de l'application hôte, choisissez Outils > Macro > Macros, sélectionnez la procédure dans la boîte de dialogue, puis cliquez sur le bouton Exécuter.

Définition

Macro or not macro ? Les macros sont des procédures `Sub` exécutables de façon autonome, donc sans arguments. Une procédure qui attend des arguments ne peut qu'être appelée par une autre et n'apparaît pas dans la liste des macros.

Vous pouvez aussi affecter un bouton de barre d'outils ou un menu de commande à une procédure `Sub`. L'activation de ce bouton ou de ce menu exécute alors la procédure. Pour plus de précisions, reportez-vous au chapitres 11.

Info

Si l'exécution d'un programme retourne une erreur, reportez-vous au chapitres 10.

Aide à l'écriture de code

Visual Basic Editor met à votre disposition des outils d'aide à l'écriture de code. Pour les activer ou désactiver, choisissez la commande Options du menu Outils et activez l'onglet Éditeur. Cochez ou décochez ensuite les options de la zone Paramètres du code (voir figure 5-18). Si vous débutez dans la programmation en VBA, activez les options d'aide à l'écriture de code ; elles vous accompagneront dans votre apprentissage.

Vérification automatique de la syntaxe

Cette option entraîne la vérification automatique de la validité de chaque ligne de code saisie. Chaque fois que vous frappez la touche Entrée ou que vous changez de ligne, la ligne en cours est vérifiée. Si une erreur est détectée, un message la décrivant s'affiche et l'instruction invalide apparaît en rouge (voir figure 5-19).

Cliquez sur le bouton OK et corrigez l'erreur si elle vous apparaît évidente, ou choisissez le bouton Aide pour afficher la rubrique associée. Celle-ci vous présente les sources probables de l'erreur et vous propose des solutions adaptées. Dès que vous entrez une modification dans l'instruction incriminée, celle-ci retrouve sa couleur normale. Si l'erreur de syntaxe n'est pas résolue, le message s'affichera de nouveau lorsque vous changerez de ligne.

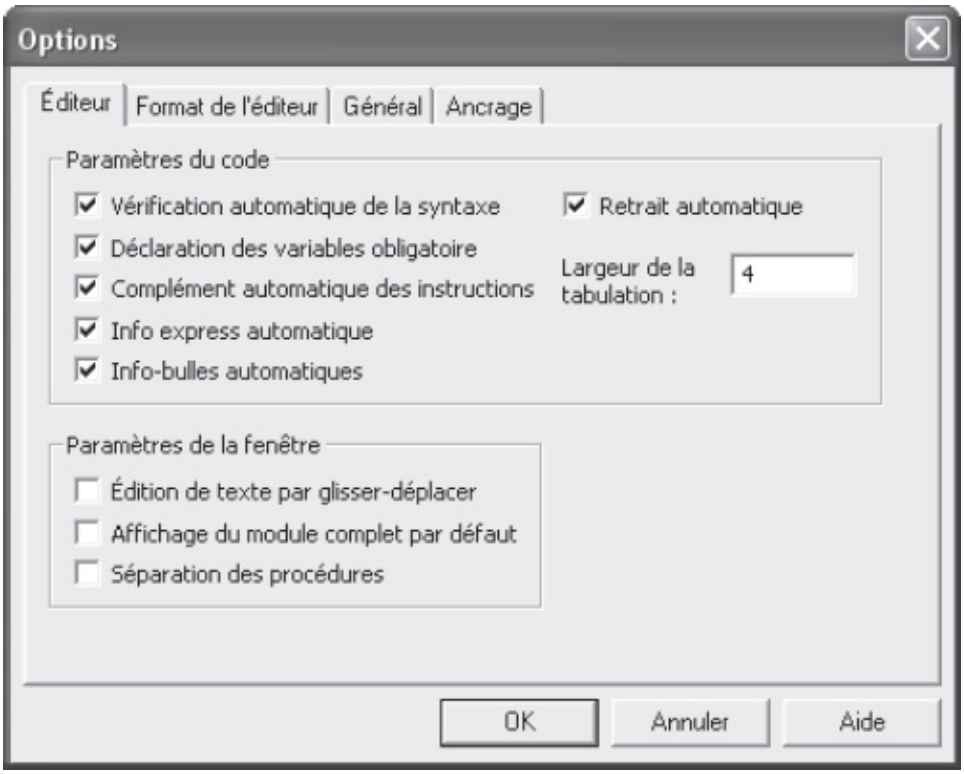


Figure 5-18 – *Visual Basic Editor propose des outils d'aide à l'écriture de code.*

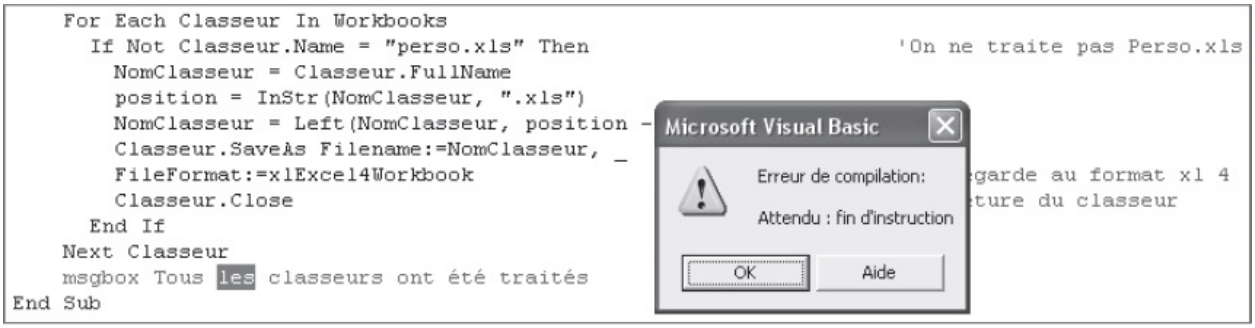


Figure 5-19 – *Vous êtes prévenu chaque fois qu'une erreur est détectée. L'instruction invalide est ici encadrée en gris.*

Complément automatique des instructions

Cette option affiche une liste alphabétique de mots-clés possibles chaque fois que l'attente d'un complément est reconnue lors de l'écriture de code. C'est par exemple le cas lorsque vous saisissez un nom de propriété appelant un objet, directement suivi d'un point. Visual Basic Editor reconnaît alors qu'un membre de l'objet (propriété ou méthode) est attendu et en affiche la liste (voir

figure 5-20).

Vous pouvez alors sélectionner l'un des éléments de la liste (souris ou touches fléchées, puis Espace pour valider), ou continuer à saisir votre code sans tenir compte de la liste affichée.

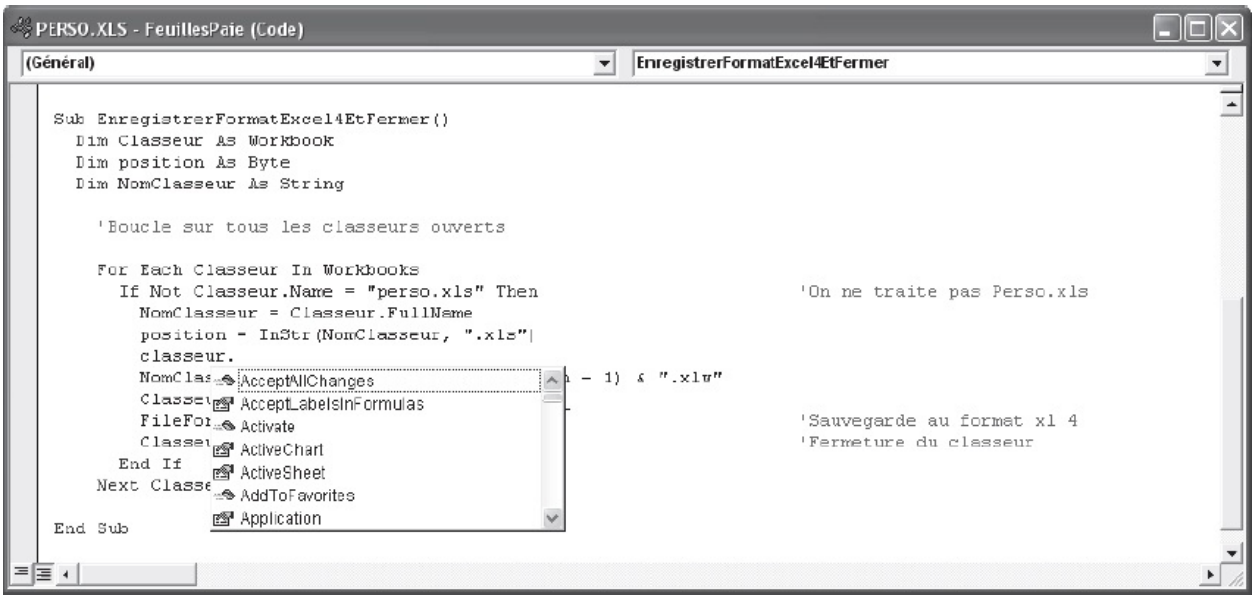


Figure 5-20 – *L'option de complément automatique des instructions affiche la liste des mots-clés possibles.*

Info express automatique

Chaque fois qu'une fonction intégrée de Visual Basic ou une procédure `Function` du module est reconnue, l'option Info express automatique en affiche la syntaxe détaillée, dans un cadre situé sous l'instruction saisie (voir figure 5-21).

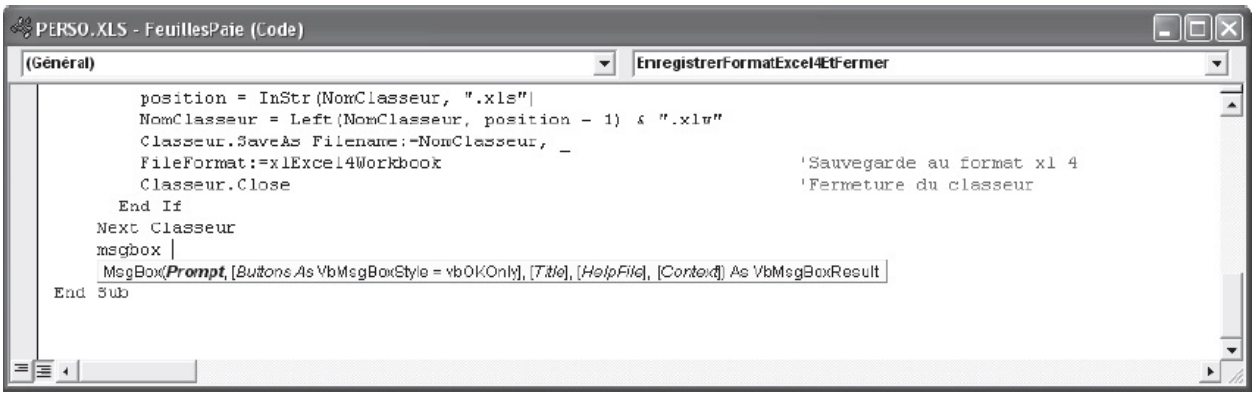


Figure 5-21 – *L'option Info express automatique affiche la syntaxe des*

fonctions au cours de la saisie.

Astuce

Les options d'aide à l'écriture de code dans Visual Basic Editor peuvent aussi être activées *via* la barre d'outils Édition.

6

Variables et constantes

Les variables sont un élément essentiel de la programmation. Elles servent à stocker les informations de votre choix à tout moment de l'exécution d'un programme, pour les réexploiter à n'importe quel autre moment. Vous pouvez, par exemple, stocker le nombre de classeurs ouverts, le nom du fichier, la valeur ou l'adresse d'une cellule, les informations entrées par l'utilisateur dans une feuille VBA, etc.

Déclarer une variable

Pour créer une variable, vous devez la déclarer, c'est-à-dire lui affecter un nom qu'il suffira par la suite de réutiliser pour exploiter la valeur qui y est stockée. La déclaration de variables en VBA peut être implicite ou explicite. Autrement dit, les programmes VBA savent reconnaître une nouvelle variable sans qu'elle soit préalablement créée dans une instruction de déclaration. Vous pouvez aussi paramétrer Visual Basic Editor afin d'exiger la déclaration explicite des variables avant leur utilisation.

Déclaration implicite

Si la déclaration explicite des variables n'est pas requise, le simple fait de faire apparaître un mot non reconnu par le programme dans une instruction d'affectation suffira pour que ce nom soit considéré comme une variable de type `Variant` – les types de variables sont présentés plus loin dans ce chapitre. C'est le cas dans l'exemple suivant :

```
Sub DéclarImpliciteDeVariables()
  MaVar = Range("D7").Value
  MsgBox "La somme totale des transactions est " & MaVar, _
         vbInformation + vbOKOnly
  Instructions
End Sub
```

Le mot `MaVar` apparaît pour la première fois dans une instruction d'affectation

valide. La variable `MaVar` est donc créée et reçoit la valeur de la cellule D7. Elle est ensuite utilisée pour afficher un message à l'attention de l'utilisateur (voir figure 6-1). L'opérateur de concaténation `&` sert à faire apparaître la valeur de la variable au cœur d'une chaîne de caractères définie.

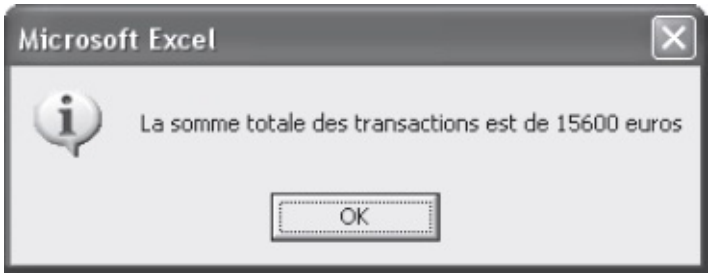


Figure 6-1 – *Les variables peuvent être utilisées dans des chaînes à l'aide de l'opérateur de concaténation.*

Vous pouvez attribuer le nom de votre choix à une variable, à condition qu'il respecte les règles suivantes :

- il doit commencer par une lettre ;
- il ne peut contenir plus de 255 caractères ;
- le point, le point d'exclamation, les espaces et les caractères @, &, $ et # ne sont pas autorisés ;
- ce nom ne doit pas être un *mot réservé*, c'est-à-dire un mot reconnu comme un élément du langage Visual Basic (nom de fonction, d'objet, de propriété, d'argument nommé, etc.).

Déclaration explicite

Il est fortement recommandé de déclarer explicitement les variables avant de les utiliser.

Forcer la déclaration des variables avec Option Explicit

Pour forcer la déclaration des variables avant leur utilisation, ajoutez l'instruction `Option Explicit`. Son utilisation évite les risques d'erreurs liées à une faute de frappe dans le nom d'une variable. Considérez la procédure suivante :

```
Sub MacroErreur()
  MaVariable = Workbooks.Count
    While MaVarable<10
      Workbooks.Add
      MaVariable = MaVariable + 1
```

```
    Wend
End Sub
```

Cette procédure a pour but d'ouvrir dix classeurs en boucle. Le nombre de classeurs ouverts (`Workbooks.Count`) est affecté à la variable `MaVariable`. Une instruction `While…Wend` – que vous découvrirez dans le prochain chapitre – est utilisée pour créer un nouveau classeur (`Workbooks.Add`) et ajouter 1 à `MaVariable` tant que la valeur de celle-ci est inférieure à 10. Cependant, le nom de la variable a été incorrectement saisi dans la condition. `MaVarable` n'existant pas, elle est créée, mais aucune valeur ne lui est affectée. La procédure ouvre donc des documents et incrémente `MaVariable` de 1 à l'infini, sans que la condition `MaVarable<10` ne soit jamais respectée.

Info

Pour interrompre une macro s'exécutant à l'infini, tapez la combinaison clavier Ctrl+Pause.

Pour éviter ce type d'erreur, forcez la déclaration explicite des variables. Pour cela, placez-vous dans la section Déclarations de la fenêtre Code du module et saisissez-y l'instruction `Option Explicit` (voir figure 6-2).

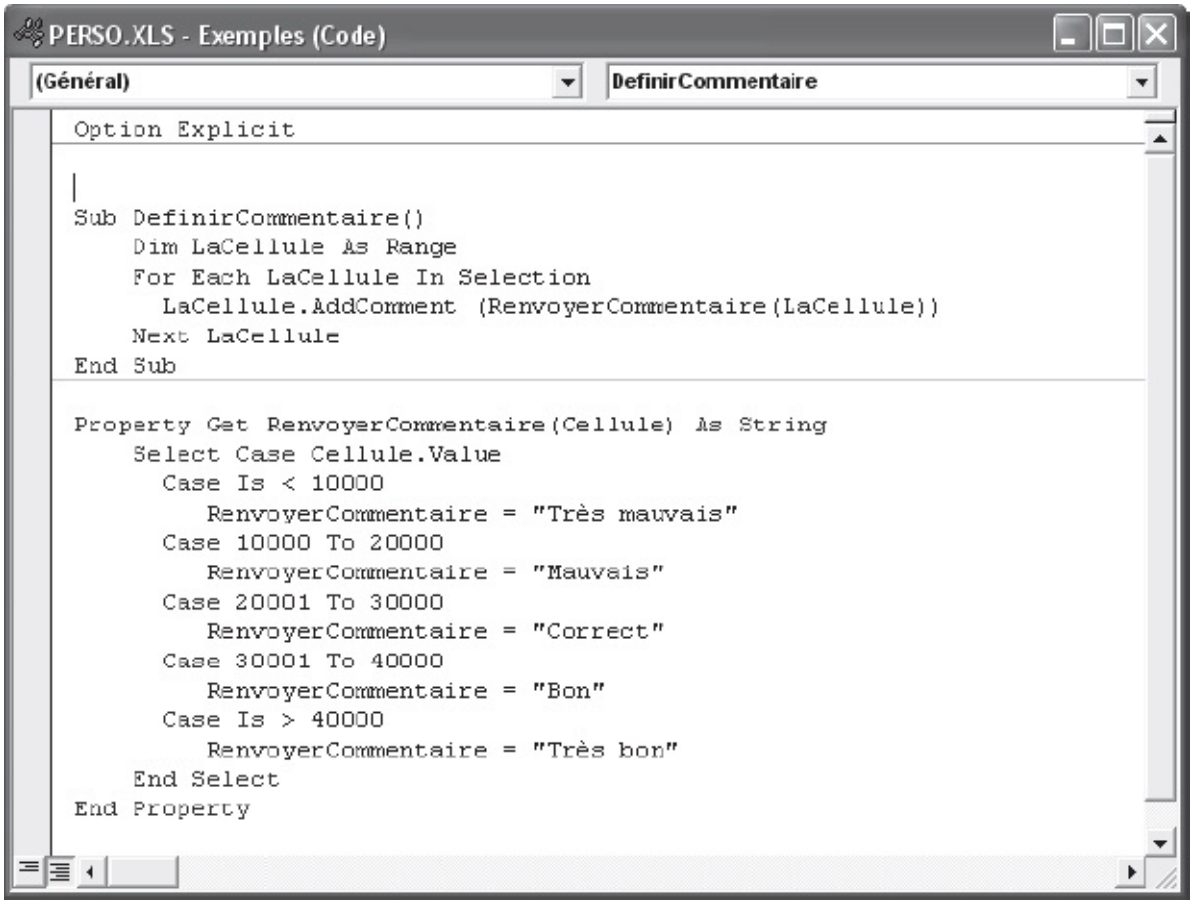


Figure 6-2 – *L'instruction Option Explicit doit se trouver dans la section Déclarations de la fenêtre Code.*

Dorénavant, l'apparition de noms de variables non préalablement déclarées à l'aide de l'instruction `Dim` générera une erreur (figure 6-3).

Figure 6-3 – *L'option Option Explicit détecte immédiatement les erreurs de saisie dans les noms de variables.*

Vous pouvez aussi paramétrer Visual Basic Editor pour que l'instruction `Option Explicit` soit systématiquement placée dans la section de Déclarations de tout nouveau module.

1. Sélectionnez la commande Options du menu Outils et placez-vous sur l'onglet Éditeur.
2. Cochez la case Déclaration explicite des variables, puis cliquez sur OK.

Déclarer une variable avec Dim

La déclaration de variables se fait à l'aide de l'instruction `Dim`, selon la syntaxe suivante :

```
Dim NomVariable As Type
```

L'argument *`Type`* est facultatif, mais la déclaration d'un type de variable fait souvent économiser de l'espace mémoire et améliore ainsi les performances de votre programme. Lors de la déclaration, une variable de type `String` prend pour valeur une chaîne vide, une variable numérique prend la valeur 0.

Dans l'exemple suivant, les variables `Message`, `Boutons` et `Titre` sont déclarées à l'aide de l'instruction `Dim`, des valeurs leur sont ensuite affectées, puis sont utilisées comme arguments de `MsgBox` afin d'afficher la boîte de dialogue de la figure 6-4.

```
Sub UtiliserDim()
  Dim Message As String
  Dim Boutons As Single
  Dim Titre As String
  Message = "La procédure est terminée."
  Boutons = vbOKOnly + vbInformation
  Titre = "C'est fini"
  MsgBox Message, Boutons, Titre
End Sub
```

Figure 6-4 – *Des variables peuvent être utilisées comme arguments d'une fonction.*

Il est possible de déclarer plusieurs variables dans une même instruction `Dim`, selon la syntaxe suivante :

```
Dim NomVar1 As Type, NomVar2 As Type, …, NomVarn As Type
```

Les trois instructions de déclaration de l'exemple précédent peuvent ainsi être ramenées à une seule :

```
Dim Message As String, Boutons As Single, Titre As String
```

Gardez à l'esprit que pour affecter un type aux variables d'une telle instruction, celui-ci doit être mentionné pour chacune des variables déclarées. L'instruction suivante déclare une variable `Message` de type `Variant` et une variable `Titre` de type `String` :

```
Dim Message, Titre As String
```

Types de données des variables

Le type d'une variable détermine la nature de l'information qui peut y être stockée. Il peut s'agir d'une valeur numérique (un nombre ou une expression renvoyant un nombre), d'une chaîne de caractères, d'une date, etc. La valeur donnée à une variable dans une instruction d'affectation doit être compatible avec son type. Par exemple, déclarer une variable de type numérique et lui affecter par la suite une chaîne de caractères générera une erreur.

Sans précision lors de sa déclaration, la variable sera de type *Variant* et acceptera tous les types de données.

Chaînes de caractères

Les variables de type *String* – encore appelées « variables de chaîne » – stockent toute expression renvoyant une valeur de type chaîne (chaîne, propriété, fonction, etc.). Utilisez la syntaxe suivante :

```
Dim NomVariable As String
```

Une chaîne doit être placée entre guillemets. Si vous souhaitez insérer des guillemets dans une chaîne (ou tout autre caractère), utilisez conjointement l'opérateur de concaténation `&` et la fonction `Chr` selon la syntaxe suivante :

```
"Chaîne de car." & Chr(codeANSI) & "Chaîne de car."
```

où *codeANSI* est le code ANSI du caractère à insérer.

> **Info**
> L'opérateur + peut aussi être utilisé. Préférez cependant l'opérateur & pour concaténer des chaînes, afin de les distinguer des additions de valeurs numériques qui, elles, requièrent l'opérateur +.

Les instructions d'affectation suivantes sont toutes valides :

- `Prénom = "Luc"`
- `NomFichier = ActiveWorkbook.Name`
- `Message = "Le nom du classeur actif est " & Chr(34) & ActiveWorkbook.Name & Chr(34) & "."`

La première instruction affecte une chaîne définie à la variable `Prénom`. La seconde affecte la valeur de la propriété `Name` du classeur actif à la variable `NomFichier`. La troisième conjugue l'affectation de chaînes définies dans le texte, renvoyées par une fonction et renvoyées par une propriété, en les concaténant à l'aide de l'opérateur `&`.

La macro suivante affiche la boîte de dialogue représentée à la figure 6-5.

```
Sub ConcatenerLesChaines()
  Dim Message As String, Boutons As Single, Titre As String
  Message = "Le nom du document actif est " & Chr(34) & _
            ActiveWorkbook.Name & Chr(34) & "."
  Boutons = vbInformation + vbOKOnly
  Titre = "Concaténation de chaînes"
  MsgBox Message, Boutons, Titre
End Sub
```

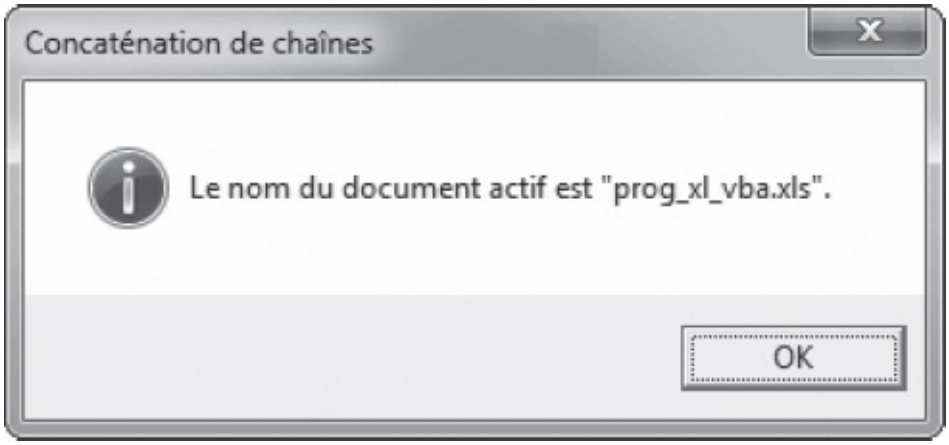


Figure 6-5 – *Le message affiché par la fonction MsgBox est toujours une chaîne de caractères.*

Les variables de chaîne définies précédemment sont dites de *longueur variable* et acceptent jusqu'à environ deux milliards de caractères. Vous pouvez cependant déclarer des variables de chaîne de *longueur fixe*, autorisant de 1 à 65 400 caractères, selon la syntaxe suivante :

```
Dim NomVariable As String * longueur
```

Les variables de longueur fixe économisent la mémoire utilisée par un programme – et donc en améliorent les performances –, mais elles doivent être utilisée prudemment. En effet, si la chaîne affectée à une variable de longueur fixe dépasse la capacité de cette dernière, elle sera purement et simplement rognée. Remplacez la déclaration de l'exemple précédent par celle-ci :

```
Dim Message As String*15, Boutons As Single, Titre As String*5
```

Vous obtenez la boîte de dialogue présentée à la figure 6-6.

Figure 6-6 – *Utilisez les variables de chaîne de longueur fixe avec prudence.*

Valeurs numériques

Les variables numériques stockent des valeurs sur lesquelles vous pouvez effectuer des opérations arithmétiques. Il existe plusieurs types de variables numériques (tableau 6-1). Elles se distinguent par l'échelle des valeurs qu'elles acceptent et par la place qu'elles occupent en mémoire.

Tableau 6-1. Types de données numériques

Types de données	Valeurs acceptées	Mémoire occupée
`Byte` (octet)	Nombre entier, compris entre 0 et 255	1 octet
`Integer` (entier)	Nombre entier compris entre –32 768 et 32 767	2 octets
`Long` (entier long)	Nombre entier compris entre –2 147 483 648 et 2 147 483 647	4 octets
`Single` (simple précision)	Nombre à virgule flottante compris entre –1,401298E-45 et –3,402823E38 ou entre 1,401298E-45 et 3,402823E38	4 octets
`Double` (double précision)	Nombre à virgule flottante compris entre –1,79769373486232E308 et –4,94065645841247E-324 ou entre 4,94065645841247E-324 et 1,79769373486232E308	8 octets
`Currency` (monétaire)	Nombre à virgule fixe, avec quinze chiffres pour la partie entière et quatre chiffres pour la partie décimale, compris entre –922 337 203 685 477,5808 et 922 337 203 685 477,5807	8 octets

Si votre programme exploite beaucoup de variables, l'affectation à chacune du type approprié (celui qui exploite le moins d'espace mémoire) en améliorera les performances. Veillez cependant à ce que les valeurs qu'une variable est susceptible de prendre soient toujours couvertes par le type accepté par la variable. Par exemple, si une valeur supérieure à 255 ou inférieure à 0 est affectée à une variable de type `Byte`, une erreur sera générée (voir figure 6-7).

Figure 6-7 – *Le type d'une variable doit couvrir l'ensemble des valeurs possibles lors de l'exécution du programme.*

Attention

Utilisez le point comme séparateur décimal dans le code VBA. L'utilisation de la virgule génère une erreur.

Une variable numérique peut être concaténée avec une chaîne de caractères à l'aide de l'opérateur `&`. Il est aussi possible de l'affecter à toute expression renvoyant une valeur numérique (chaîne, propriété, fonction, etc.) et d'effectuer des opérations à l'aide des opérateurs arithmétiques présentés dans le tableau 6-2.

Tableau 6-2. Les opérateurs arithmétiques

Opérateur	Description
+	Addition
-	Soustraction
*	Multiplication
/	Division
\	Division. Seule la partie entière du résultat est renvoyée (l'opération 18\5 retournera la valeur 3).
^	Élévation à la puissance (2^4 renvoie 16).

À condition que le type de la variable numérique couvre les valeurs qui lui sont affectées, les instructions suivantes sont toutes valides :

- `MaValeur = 58`
- `NbreClasseur = Workbooks.Count`
- `Range("D5").Value = (Range("D3").Value + Range("D4").Value) / 2`
- `SurfaceCercle = (varRayon^2) * 3.14`

La première instruction affecte une valeur définie à la variable `MaValeur`. La deuxième affecte la valeur de la propriété `Count` de l'objet (la collection) `Workbooks` (le nombre de classeurs ouverts). La troisième utilise l'opérateur `+` pour additionner les valeurs des cellules D3 et D4, puis l'opérateur `/` pour diviser la valeur obtenue par deux. La dernière instruction élève au carré la variable `varRayon` à l'aide de l'opérateur `^`, puis multiplie le résultat par 3,14 à l'aide de l'opérateur `*`.

En revanche, l'instruction suivante affecte une valeur de type chaîne à la variable numérique `MaVar` et génère une erreur (voir figure 6-8).

```
Sub ErreurAffectation()
  Dim MaVar As Byte
  MaVar = ActiveWorkbook.Name
End Sub
```

Figure 6-8 – *Une chaîne de caractères ne peut être affectée à une variable numérique.*

Rappelez-vous que les constantes sont en réalité des valeurs numériques. Une constante peut donc être affectée à une variable numérique et entrer dans une expression arithmétique. Veillez à ne pas utiliser l'opérateur `&`, réservé à la concaténation de chaînes. Par exemple, l'instruction :

```
MsgBox "Le message de la bdg", vbOKOnly + vbInformation, "Titre"
```

est valide, tandis que l'instruction :

```
MsgBox "Le message de la bdg", vbOKOnly & vbInformation, "Titre"
```

générera un message d'erreur, car l'opérateur `&` est utilisé pour additionner les valeurs affectées aux constantes `vbOKOnly` et `vbInformation`.

VBA intègre de nombreuses fonctions pour manipuler les valeurs numériques. Par exemple, `ABS(`*`nombre`*`)` renvoie la valeur absolue du *`nombre`* spécifié entre parenthèses et `Int(`*`nombre`*`)` renvoie la partie entière.

La liste des fonctions VBA est consultable dans l'aide en ligne. Choisissez la rubrique Référence du langage Visual Basic. Vous y trouverez une rubrique Fonctions répertoriant les fonctions par ordre alphabétique. Vous pouvez également y choisir la rubrique Liste et index. Vous y découvrirez des rubriques thématiques, telles que Résumé des mots-clés financiers, ou Résumé des mots-clés mathématiques.

Valeurs booléennes

Une variable de type `Boolean` sert à stocker le résultat d'une expression logique. Elle renvoie la valeur `True` ou `False` et occupe deux octets en mémoire. Elle peut aussi se voir affecter une valeur numérique : si cette dernière est égale à zéro, la variable prendra la valeur `False`, la valeur `True` dans le cas contraire. Une variable booléenne utilisée comme chaîne de caractères renvoie le mot Vrai ou Faux.

Dans l'exemple suivant, `ClasseurSauvegardé` prend la valeur `False` si le classeur actif a subi des modifications depuis le dernier enregistrement ; `True` dans le cas contraire. Une boîte de dialogue affiche ensuite le message Vrai ou Faux, en fonction de la valeur de `ClasseurSauvegardé`.

```
Sub ClasseurSauvegardéOuNon()
  Dim ClasseurSauvegardé As Boolean
  ClasseurSauvegardé = ActiveWorkbook.Saved
  MsgBox ClasseurSauvegardé
End Sub
```

Dates

Les dates comprises entre le 1er janvier 100 et le 31 décembre 9999 peuvent être affectées à des variables de type `Date`. Ces dernières sont stockées sous forme de nombres à virgule flottante et occupent huit octets en mémoire.

La partie entière du nombre représente le jour. 0 correspond au 30 décembre 1899, 1 au 31 décembre 1899, etc. Les valeurs négatives représentent des dates antérieures au 30 décembre 1899. La partie décimale du nombre représente

l'heure. 0.5 correspond à 12 heures, 0.75 à 18 heures, etc. Ainsi, 36526.00001 correspond au 1er janvier 2000, à 00:00:01.

Visual Basic intègre des fonctions pour manipuler les dates. Par exemple, `Date`, `Time` et `Now` renvoient respectivement la date courante, l'heure courante et la date courante suivie de l'heure courante. La procédure suivante affiche la boîte de dialogue de la figure 6-9 :

```
Sub DateEtHeure()
  Dim LaDate As Date, LHeure As Date
  LaDate = Date
  LHeure = Time
  MsgBox "Nous sommes le " & LaDate & ", il est " & LHeure & ".", _
         vbOKOnly + vbInformation, "Fonctions Date et Time"
End Sub
```

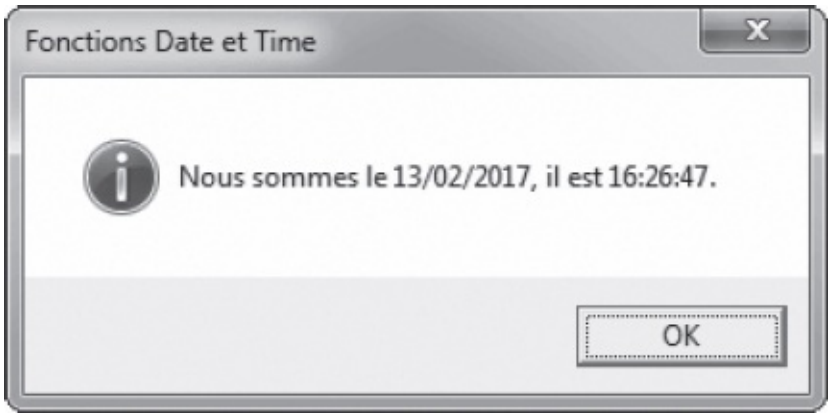


Figure 6-9 – *Visual Basic intègre des fonctions pour manipuler les dates et les heures.*

Pour plus d'informations sur les fonctions de date et d'heure, reportez-vous à l'aide Visual Basic, en choisissant Résumé des mots-clés de date et d'heure de la rubrique Index/Listes du Manuel de référence du langage. Vous pouvez également vous reporter au Manuel de référence pour Microsoft Excel, disponible dans le sommaire de l'aide.

Attention

L'installation par défaut d'Excel n'installe pas l'aide en ligne de VBA. Vous devrez lancer de nouveau l'installation en l'incluant.

Type Variant

Une variable de type `Variant` accepte des valeurs de tout type et peut être automatiquement convertie d'un type à l'autre, tant que sa valeur est

compatible. Autrement dit, une variable de type `Variant` peut être initialement une chaîne de caractères, qui sera exploitée par la suite en tant que valeur numérique. Si les données qui lui sont affectées sont assimilables à une valeur numérique, la conversion se fera automatiquement, lorsque l'instruction assimilable à une opération numérique sera exécutée.

Si les variables de type `Variant` sont très pratiques, elles occupent un espace en mémoire plus important que les autres types et peuvent donc ralentir l'exécution du programme.

Une variable, une constante ou un argument dont le type n'est pas déclaré est par défaut de type `Variant`.

Les instructions suivantes sont équivalentes :

```
Dim MaVar
Dim MaVar As Variant
```

Variables de matrice

Une variable de matrice ou de type `Array`, encore appelée tableau, est capable de stocker plusieurs valeurs de même type, contrairement à une variable ordinaire ne pouvant recevoir qu'une seule valeur. Vous pouvez par exemple y stocker les chiffres d'affaires de tous les représentants. Pour déclarer une variable de matrice, utilisez la syntaxe suivante :

```
Dim NomVariable(NbreElements) As Type
```

Pour affecter des valeurs à une variable de matrice ou accéder à ces dernières, il suffit de spécifier la position de la valeur stockée dans la variable. La première valeur recevant l'index 0, la dernière valeur est toujours égale au nombre d'éléments contenus dans la variable moins 1.

Astuce

Pour démarrer l'index d'un tableau à 1 plutôt qu'à 0, placez l'instruction `Option Base 1` dans la section Déclarations du module.

Une variable de matrice peut également être déclarée selon la syntaxe suivante :

```
Dim NomVariable(Début To Fin) As Type
```

où *`Début`* et *`Fin`* définissent la plage de valeurs qui sera utilisée pour stocker et accéder aux données de la variable.

Si une variable de matrice sert à stocker des données de même type, il est alors

recommandé de déclarer un type approprié. Dans l'exemple suivant, `JoursSemaine` stocke sous forme de chaînes les jours de la semaine. Une structure de contrôle `For…Next` est ensuite utilisée pour afficher dans une boîte de dialogue les valeurs contenues par la variable.

```
Sub VarMatrice()
  Dim JoursSemaine(7) As String
  JoursSemaine(0) = "Lundi"
  JoursSemaine(1) = "Mardi"
  JoursSemaine(2) = "Mercredi"
  JoursSemaine(3) = "Jeudi"
  JoursSemaine(4) = "Vendredi"
  JoursSemaine(5) = "Samedi"
  JoursSemaine(6) = "Dimanche"
  Dim compteur as Byte
    For compteur = 0 To 6
      MsgBox JoursSemaine(compteur)
    Next compteur
End Sub
```

Une variable de matrice peut aussi stocker des données de types différents. Elle doit alors être de type `Variant`. La procédure suivante stocke dans une seule variable le nom, la date de naissance, l'adresse, la fonction et le salaire d'un employé. Ces données sont ensuite affichées dans une boîte de dialogue (voir figure 6-10).

```
Sub InfosEmployé
  Dim Employé(1 To 5) As Variant
  Employé(1) = "Jean Dupont"
  Employé(2) = "25/12/71"
  Employé(3) = "14, rue des Arts"
  Employé(4) = "Chargé d'études"
  Employé(5) = 2000
  MsgBox Employé(1) & " est né le " & Employé(2) & ". Il habite " & _
  Employé(3) & " et est " & Employé(4) & ". Son salaire est de : " & _
  Employé(5) & " euros", vbOKOnly + vbInformation, "Infos employé"
End Sub
```

Astuce

Utilisez la fonction `IsArray` pour vérifier si une variable est de type `Array`.

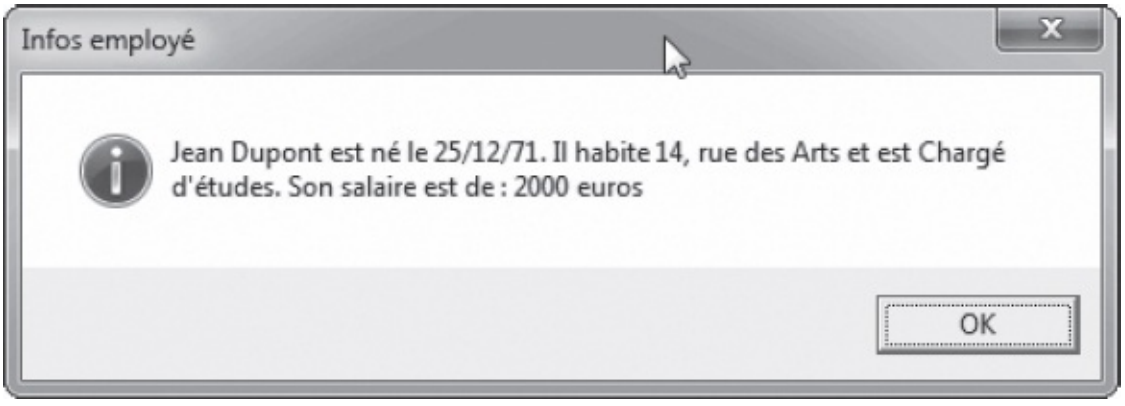


Figure 6-10 – *Toutes les informations relatives à un même sujet peuvent être stockées dans une seule variable de matrice.*

Variables de matrice multidimensionnelles

Les variables de matrice peuvent être multidimensionnelles. Les données sont alors stockées horizontalement et verticalement, à la manière d'un tableau. Ces variables servent à stocker de façon cohérente les valeurs d'une feuille Excel. Elles se déclarent selon la même syntaxe, en ajoutant simplement les arguments d'index de début et de fin pour la deuxième dimension :

```
Dim NomVariable(Début1 To Fin1, Début2 To Fin2) As Type
```

Considérez la feuille de classeur représentée à la figure 6-11. Elle représente les ventes mensuelles, pour l'année 2003, de quatre types de produits – soit 12 lignes sur quatre colonnes. Une variable de matrice multidimensionnelle stockera l'ensemble de ces valeurs :

```
Dim MonTableau(1 To 12, 1 To 4) As Single
```

	A	B	C	D	E	F
1		Livres	Vidéo	Hi-Fi	Autres	
2	Janvier	58 963,00	45 225,00	85 485,00	45 225,00	
3	Février	45 895,00	32 568,00	79 658,00	32 568,00	
4	Mars	69 785,00	46 895,00	25 689,00	46 895,00	
5	Avril	45 214,00	54 897,00	49 652,00	54 897,00	
6	Mai	45 258,00	32 568,00	36 550,00	32 568,00	
7	Juin	38 652,00	97 632,00	56 320,00	97 632,00	
8	Juillet	32 510,00	45 863,00	45 520,00	45 863,00	
9	Août	28 952,00	32 568,00	47 965,00	32 568,00	
10	Septembre	45 693,00	94 625,00	29 865,00	94 625,00	
11	Octobre	48 956,00	31 582,00	16 495,00	31 582,00	
12	Novembre	65 920,00	21 458,00	75 632,00	21 458,00	
13	Décembre	95 120,00	12 589,00	12 589,00	12 589,00	
14						

Figure 6-11 – *Utilisez une variable de matrice multidimensionnelle pour stocker les valeurs d'un tableau.*

Il suffit ensuite d'affecter logiquement les valeurs de la feuille aux espaces de stockage de la variable :

```
MonTableau(1,1) = Cells(2,2).Value
MonTableau(1,2) = Cells(2,3).Value
MonTableau(1,3) = Cells(2,4).Value
MonTableau(1,4) = Cells(2,5).Value
MonTableau(2,1) = Cells(3,2).Value
```

```
MonTableau(2,2) = Cells(3,3).Value
MonTableau(2,3) = Cells(3,4).Value
MonTableau(2,4) = Cells(3,5).Value
Etc.
```

Ainsi, pour accéder aux ventes d'un mois, il suffira de spécifier la valeur correspondante comme premier index de la variable `MonTableau` (1 = janvier, 2 = février, etc.). De manière similaire, la catégorie de ventes correspond à une valeur du second index (1 = Livres, 2 = Vidéo, 3 = Hi-Fi, 4 = Autres). Par exemple, `MonTableau(1,1)` renverra les ventes de janvier pour les livres et `MonTableau(12,2)` renverra les ventes de décembre pour la vidéo.

N'hésitez pas à utiliser les variables de matrice pour stocker les données d'une feuille Excel auxquelles un programme VBA doit accéder à de multiples reprises. La variable ainsi créée est chargée en mémoire. L'accès aux données qu'elle contient est nettement plus rapide qu'un accès aux valeurs contenues dans les cellules d'une feuille de calcul.

L'utilisation d'une structure de contrôle `For…Next` servira à affecter l'ensemble des valeurs à une variable de matrice en quelques lignes de code. Vous apprendrez à utiliser cette structure au chapitres 7.

Info

Une variable de matrice n'est pas limitée à deux dimensions. Vous pouvez parfaitement en créer une à trois dimensions, ou plus.

Conseil

La fonction `LBound` (resp. `UBound`) renvoie le plus petit (resp. le plus grand) indice disponible pour une dimension spécifiée d'un tableau :

`LBound(`*`NomVariable`*`, `*`Dimension`*`)` et `UBound(`*`NomVariable`*`, `*`Dimension`*`)`

Si l'argument *`Dimension`* est omis, le plus petit ou le plus grand indice de la première dimension est renvoyé.

Variables de matrice dynamiques

Si vous ne spécifiez pas de valeur de taille entre les parenthèses qui suivent le nom de la variable de matrice, celle-ci sera dynamique :

```
Dim NomVariable()
```

Avant d'affecter des valeurs à la variable ainsi créée, vous devrez la redimensionner à l'aide de l'instruction `ReDim`, selon la syntaxe suivante :

```
ReDim NomVariable(Début To Fin)
```

Vous pouvez utiliser le mot-clé `ReDim` pour redimensionner une variable de

matrice autant de fois que vous le souhaitez. Ces variables sont intéressantes lorsque vous ne connaissez pas *a priori* la quantité de données à stocker. Supposez que, dans l'exemple précédent, la feuille de calcul des ventes ne soit pas annuelle, mais mensuelle. Le tableau s'enrichirait alors tous les mois d'une nouvelle ligne. Pour que votre programme fonctionne tout au long de l'année, vous devrez créer une variable de matrice de longueur variable :

```
1: Sub AffectationVariableArray()
2:     Dim MonTableau() As Single
3:     Dim DerniereLigne As Byte
4:     DerniereLigne = Range("A2").End(xlDown).Row
5:     Dim NbreDeLignes As Byte
6:     NbreDeLignes = DerniereLigne - 1
7:     ReDim MonTableau(NbreDeLignes,4)
8:     Instructions d'affectation de valeurs à MonTableau
9: End Sub
```

Aux lignes 2 et 3, les variables `MonTableau` et `DerniereLigne` sont déclarées. L'instruction de la ligne 4 sert à affecter à `DerniereLigne` le numéro de la dernière ligne contenant des données. La fonction `End` renvoie l'objet `Range` correspondant à la dernière cellule non vide sous (`xlDown`) la cellule A2. La propriété `Row` renvoie le numéro de ligne de cet objet. La variable `NbreDeLignes` est créée ligne 5. On lui affecte ensuite une valeur égale à `DerniereLigne` - 1, soit le nombre de lignes contenant des données à stocker dans la variable (la première ligne ne contenant que des intitulés de colonnes). À la ligne 7, `MonTableau` est redimensionnée de façon à accueillir l'ensemble des chiffres de ventes de la feuille.

Attention

Lorsque vous redimensionnez une variable de matrice, celle-ci est réinitialisée et toutes les valeurs qui y étaient stockées sont perdues. Pour les conserver, placez le mot-clé `Preserve` devant l'instruction `ReDim`. L'utilisation de ce mot-clé est cependant subordonnée à certaines conditions :

- Vous ne pouvez redimensionner que la dernière dimension de la variable.
- Vous ne pouvez pas modifier le nombre de dimensions du tableau.
- Vous ne pouvez qu'agrandir le tableau. Si vous le réduisez, toutes les données seront perdues.

La fonction première du tableur étant d'effectuer des calculs sur des données affichées sous forme de tableaux, les variables de matrice sont très utilisées dans les programmes VBA pour Excel. En effet, en stockant les données de feuilles de calcul sous forme de variables, vous améliorez sensiblement les performances du programme.

Variables objets

Ces variables sont utilisées pour faire référence à un objet et occupent 4 octets en mémoire. Une fois une variable objet définie, vous pouvez en chercher ou définir les propriétés, ou encore exécuter l'une de ses méthodes en faisant simplement référence à la variable. Utilisez la syntaxe suivante :

```
Dim NomVariable As Object
```

Il est mieux de remplacer `Object` par un nom d'objet reconnu par l'application. Vous pouvez, par exemple, déclarer une variable objet `Workbook` (classeur) selon la syntaxe suivante :

```
Dim MonObjetClasseur As Workbook
```

Une fois la variable déclarée, vous devez lui affecter un objet précis ; utilisez pour cela le mot-clé `Set`, selon la syntaxe suivante :

```
Set NomVariable = Expression
```

où *`Expression`* renvoie un objet de l'application. Dans l'exemple suivant, la variable `Police` est déclarée en tant qu'objet `Font`, puis se voit affecter l'expression `Workbooks("Representant.xlsx").Sheets("Feuil1").Range("A1:D5").Font`, soit l'objet `Font` (police) de la plage de cellules A1:D5 de la feuille libellée Feuil1 du classeur Representant.xlsx. La variable est ensuite utilisée pour définir la propriété `Bold` de l'objet à `True`, c'est-à-dire pour affecter l'attribut gras à la plage de cellules A1:D5 de ce classeur.

```
Sub VariablesObjet()
  Dim Police As Font
  Set Police = _
    Workbooks("Representant.xlsx").Sheets("Feuil1").Range("A1:D5").Font
  Police.Bold = True
End Sub
```

La fonction GetObject

Ce type de variable vous permet d'agir sur un objet sans que celui-ci soit ouvert. Ainsi, il est possible de chercher ou modifier les valeurs d'une feuille Excel sans que le fichier ne soit ouvert. Pour accéder à un objet, stockez-le dans une variable et utilisez la fonction `GetObject`, selon la syntaxe suivante :

```
Set MonObjet = GetObject(pathname, class)
```

où *`pathname`* et *`class`* sont des arguments nommés de type chaîne, correspondant respectivement au chemin d'accès complet au fichier et à la classe de l'objet. Si *`pathname`* est spécifié, *`class`* peut être omis.

La procédure suivante crée une variable objet de type `Workbook` et lui affecte le fichier Representant.xlsx, situé sur le Bureau de Windows.

```
Sub AccederObjetFerme()
  Dim ObjetClasseur As Workbook
  Set ObjetClasseur = _
    GetObject("C:\Users\Nom_utilisateur\Desktop\Representant.xlsx")
End Sub
```

La fonction `GetObject` est particulièrement intéressante si des données entrées dans un classeur doivent être répercutées dans un ou plusieurs autres. Vous pouvez par exemple créer un programme VBA afin que, lorsqu'un client vous passe une commande, le classeur contenant les données du stock soit mis à jour. Si nécessaire, un message s'affichera pour prévenir l'utilisateur qu'il est temps de renouveler le stock, sans même qu'il sache qu'il existe un classeur des stocks. C'est ce que fait la procédure suivante, en supposant que la valeur du stock pour le produit commandé se trouve dans la cellule A13 du classeur stock.xlsx.

```
 1: Sub Commande()
 2:   'Instructions
 3:   Dim StockRestant As Integer
 4:   Dim UnitésCommandées As Integer
 5:   UnitésCommandées = 50
 6:   StockRestant = VerifierEtMettreAJourStock(UnitésCommandées)
 7:   If StockRestant<0 Then
 8:      MsgBox "Le stock ne permet pas d'assurer la commande. " & _
          "Le stock pour ce produit est de " & _
          (StockRestant + UnitésCommandées) & " unités."
 9:      Exit Sub
10:   Else
11:      MsgBox "Commande effectuée. Le stock restant pour ce " & _
          "produit est de " & StockRestant & " unités."
12:   End If
13:   'Suite des instructions de la commande
14: End Sub


15: Function VerifierEtMettreAJourStock(QteCommande)
16:   Dim ObjetStock As Workbook
17:   Dim StockDispo As Integer
18:   Set ObjetStock = GetObject("C:\Users\Nom_utilisateur\Desktop\Stock.xlsx")
19:   StockDispo = ObjetStock.Sheets(1).Range("A13").Value
20:   VerifierEtMettreAJourStock = StockDispo - QteCommande
21:   If VerifierEtMettreAJourStock>=0 Then
22:      ObjetStock.Sheets(1).Range("A13").Value = _
          StockDispo - QteCommande
23:      ObjetStock.Save
24:   End If
25: End Function
```

Attention

Veillez à personnaliser le chemin précisé pour la fonction `GetObject` à la ligne 18, sinon cette macro ne fonctionnera pas.

À la ligne 6, la procédure `Commande` appelle la fonction `VerifierEtMettreAJourStock` en lui passant la valeur de la variable `UnitésCommandées`. La valeur 50 a été affectée à

cette variable à la ligne 5 pour faire fonctionner le programme. Il va de soi que cette variable doit être affectée au nombre d'unités réellement commandées.

`VerifierEtMettreAJourStock` déclare les variables `ObjetStock` – de type `Workbook` (ligne 16) – et `StockDispo`, de type `Integer` (ligne 17). Ligne 18, la variable `ObjetStock` se voit affecter le classeur Stock.xlsx situé sur le Bureau de Windows. À la ligne suivante, `StockDispo` reçoit la valeur de la cellule A13 de la première feuille de ce classeur. La fonction `VerifierEtMettreAJourStock` se voit ensuite affecter la valeur de `StockDispo - QteCommande`. Enfin, lignes 21 à 24, une structure `If…Then` est utilisée pour mettre à jour la valeur du stock restant. L'instruction de la ligne 21 vérifie que le stock restant après commande est supérieur à zéro. Si c'est le cas, la valeur de la cellule A13 est mise à jour en conséquence (ligne 22) et le classeur est ensuite sauvegardé (ligne 23).

La procédure `Commande` reprend alors la main. Lignes 7 à 12, une structure conditionnelle `If…Then…Else` affiche un message à l'attention de l'utilisateur, afin de l'informer sur l'état du stock. Si ce dernier est insuffisant pour assurer la commande (`StockRestant<0`), l'instruction de la ligne 8 est exécutée : elle avertit l'utilisateur que la commande n'a pu être validée et lui indique le stock disponible (`StockRestant + UnitésCommandées`). Si la commande est validée, l'instruction de la ligne 11 affiche le stock restant.

La fonction CreateObject

La fonction `CreateObject` sert à créer une instance d'objet et s'utilise selon la syntaxe suivante :

```
CreateObject(class,servername)
```

où *`class`* (classe de l'objet dont on crée une instance) et *`servername`* (facultatif, nom d'un serveur distant sur lequel est créé l'objet) sont des arguments nommés de type chaîne. Quand l'instance est créée, on accède à ses propriétés et méthodes en utilisant le nom de la variable.

Dans l'exemple suivant, une instance de l'objet Excel est créée. Un nouveau classeur est alors créé, configuré puis enregistré dans cette instance.

```
1: Sub CreerInstancesExcel()
2:    'déclaration des variables
3:    Dim Xl As Excel.Application
4:    Dim NouvClasseur As Excel.Workbook
5:    Dim NomFichier As String
6:    'création d'une instance de l'objet Excel
7:    Set Xl = CreateObject("Excel.Application")
8:    'affichage de l'objet Xl
9:    Xl.Application.Visible = True
```

```
10:    'création d'un nouveau classeur dans l'objet Xl
11:    Set NouvClasseur = Xl.Workbooks.Add
12:    'ajout d'une feuille de calcul
13:    NouvClasseur.Sheets.Add
14:    'affectation de noms aux feuilles 1 et 2
15:    NouvClasseur.Sheets(1).Name = "Quantites"
16:    NouvClasseur.Sheets(2).Name = "Chiffres"
17:    'définition du nom du classeur
18:    Dim compteur As Byte
19:    Dim Pos As Long
20:    NomFichier = "Ventes " & Date & ".xlsx"
21:    For compteur = 1 To 2
22:       Pos = InStr(NomFichier, "/")
23:       NomFichier = Left(NomFichier, Pos - 1) & "-" & _
24:                 Right(NomFichier, Len(NomFichier) - Pos)
25:    Next compteur
26:    'enregistrement du classeur
27:    NouvClasseur.SaveAs "C:\Users\Nom_utilisateur\Desktop\"
         ➥& NomFichier
28:    NouvClasseur.Close
29:    Xl.Quit
30: End Sub
```

Attention

Veillez à personnaliser le chemin précisé pour la fonction `GetObject` à la ligne 27, sinon cette macro ne fonctionnera pas.

Lignes 2 à 5, les variables sont déclarées : `Xl` et `NouvClasseur` sont des objets de type `Excel` et `Workbook` ; `NomFichier` servira à stocker le nom d'enregistrement du classeur. Ligne 7, une instance de l'objet `Application` d'Excel est créée à l'aide de l'instruction `CreateObject`, puis affectée à la variable Xl. Sa propriété `Visible` est ensuite définie à True afin de faire apparaître la session Excel à l'écran.

Ligne 11, un nouveau classeur est ajouté à l'objet `Application` et affecté à la variable objet `NouvClasseur`. Notez que, par défaut, un nouveau classeur est créé dans la session Excel à partir de laquelle le programme est exécuté. Pour qu'il le soit dans la nouvelle session, il est indispensable de faire référence à l'objet `Xl` dans l'instruction de la ligne 11. Lignes 13 à 16, une feuille est ajoutée et les deux feuilles sont renommées.

Lignes 17 à 25, le nom d'enregistrement du classeur est défini. La variable `NomFichier` se voit tout d'abord affecter le nom Ventes, suivi de la date du jour (Ventes 12/08/2007, par exemple). Ce nom contient deux fois le caractère barre oblique (/), invalide dans les noms de fichier. Lignes 21 à 25, une boucle `For…Next` est utilisée pour répéter deux fois le traitement appliqué au nom du classeur afin de substituer des traits d'union aux barres obliques (les boucles sont étudiées au prochain chapitre). On utilise pour ce faire les fonctions de manipulation de chaîne `InStr`, `Left`, `Right` et `Len`. `Instr` renvoie la position du caractère / dans la chaîne `NomFichier` (ligne 22). `Len` renvoie le

nombre de caractères de `NomFichier` (ligne 24). Les fonctions `Left` et `Right` sont utilisées pour renvoyer respectivement les caractères situés à gauche et à droite des barres obliques ; un trait d'union est placé entre les deux chaînes ainsi renvoyées.

Lignes 27 et 28, le classeur est enregistré puis fermé. Ligne 29, la méthode `Quit` est appliquée à l'objet Excel, afin de fermer la session créée en début de programme.

Libérer une variable objet

Pour annuler l'affectation d'un objet à une variable, donnez-lui la valeur `Nothing` :

```
Set Police = Nothing
```

Il est important d'affecter la valeur `Nothing` à une variable objet lorsque celle-ci n'est plus utilisée par le programme. Vous libérez ainsi l'ensemble des ressources système et mémoire associées à l'objet. Dans l'exemple de programme de stock précédent, vous devrez placer cette instruction au-dessus de la ligne 24. La variable `ObjetStock` sera ainsi libérée avant que la procédure `Commandes` ne reprenne la main.

Types de données personnalisés

Le mot-clé `Type` sert à créer des types de données personnalisés, associant les types de base et capables de stocker des informations multiples. Une telle opération se révèle intéressante lorsqu'un programme doit associer de façon récurrente différents types de données.

Contrairement aux variables de matrice, `Type` permet d'associer des types de données différents.

La déclaration d'un nouveau type de données doit être placée dans la section Déclarations du module, selon la syntaxe suivante :

```
Type NomType
   Données1 As Type
   Données2 As Type
   ….
   Donnéesn As Type
End Type
```

Les noms *Données1*, …, *Donnéesn* seront employés par la suite pour affecter des valeurs aux différents espaces de stockage des variables de type *NomType*.

Dans l'exemple qui suit, un type de données `Membre` est créé, afin d'intégrer dans

une seule variable l'ensemble des informations concernant un membre d'une association donnée.

```
Type Membre
   Prénom As String
   Nom As String
   Adresse As String
   CodePostal As String
   Ville As String
   Téléphone As String
   Age As Byte
End Type
```

Vous pouvez maintenant créer une nouvelle variable de type `Membre`. Les informations qu'elle contient seront ensuite recherchées ou définies en faisant suivre le nom de la variable d'un point, puis du nom de la donnée (voir figure 6-12) :

```
Sub NouveauMembre
  Dim NouvMembre As Membre
    With NouvMembre
      .Prénom = "Hélène"
      .Nom = "Bienvenue"
      .Adresse = "4, rue des oiseaux"
      .CodePostal = "56000"
      .Ville = "Vannes"
      .Téléphone = "00 01 02 03 04"
      .Age = 2
    End With
  MsgBox "Le nouveau membre s'appelle " & NouvMembre.Prénom & " " & _
        NouvMembre.Nom, vbOKOnly + vbInformation, "Nouveau membre"
End Sub
```

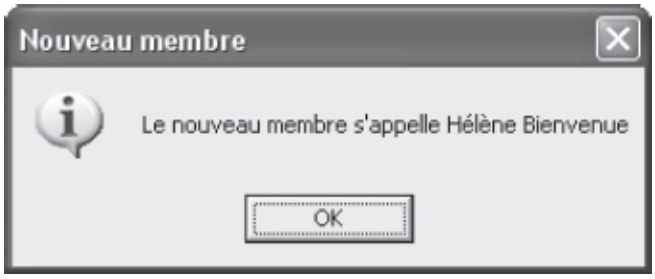


Figure 6-12 – *Les variables de type personnalisé contiennent autant d'informations que vous le souhaitez.*

Notez que, en phase de création, un complément automatique s'affiche lorsque vous faites référence à une variable de type personnalisé (voir figure 6-13).

Constantes

Les constantes attribuent un nom à une valeur fixe de n'importe quel type. Il est ainsi plus aisé d'exploiter cette valeur dans le code en faisant référence au nom

de la constante, plutôt qu'à la valeur elle-même. Par ailleurs, si une valeur est susceptible d'être modifiée (TVA, par exemple), son affectation à une constante simplifiera les éventuelles mises à jour ; il vous suffira en effet de modifier la constante en un seul endroit, plutôt que de modifier chaque occurrence de la valeur dans l'ensemble de vos projets.

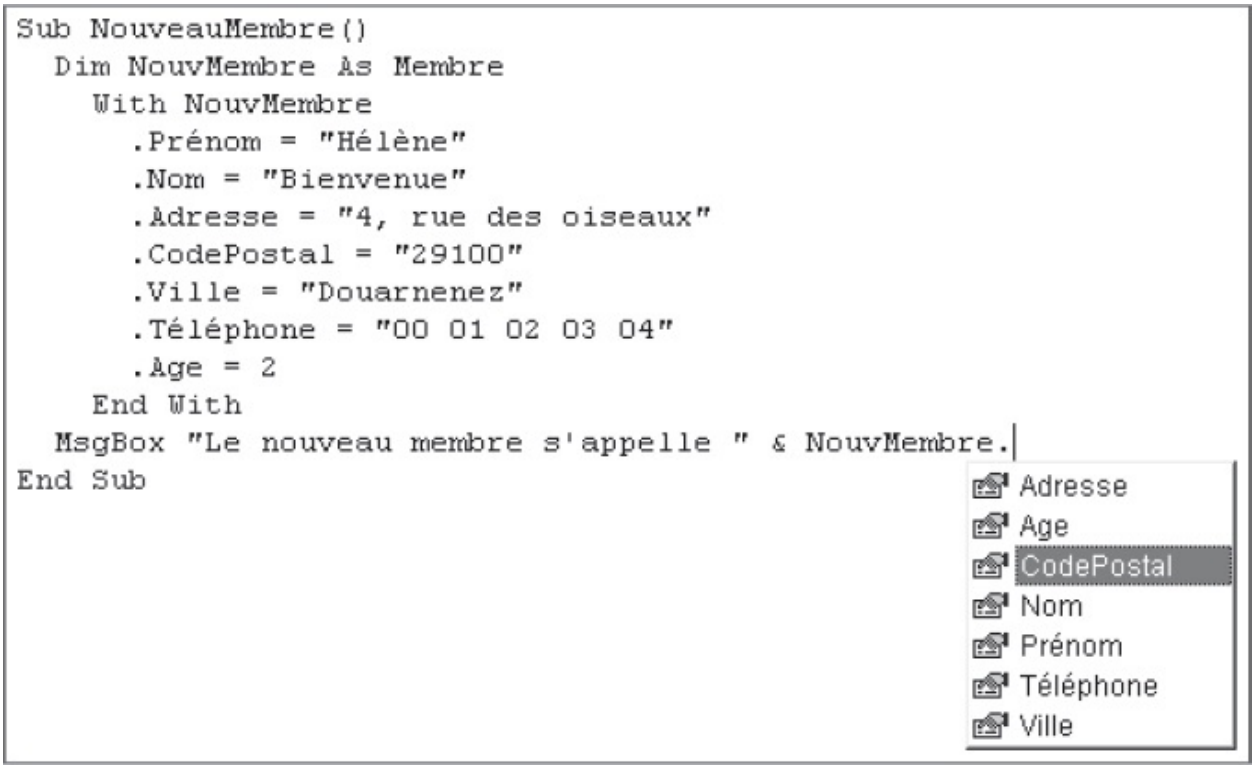


Figure 6-13 – *Le complément automatique d'instruction s'affiche pour les types de données personnalisés.*

Attention

Une fois qu'une valeur a été affectée à une constante, celle-ci ne peut être modifiée par la suite.

Utilisez l'instruction `Const`, selon la syntaxe suivante :

```
Const NomConstante As Type = Valeur
```

Par exemple, l'instruction suivante déclare la constante TVA, à laquelle la valeur 20.6 est affectée.

```
Const TVA As Single = 20.6
```

Validation et conversion des types de données

Il est souvent nécessaire de vérifier que le type des données entrées par l'utilisateur dans une cellule ou dans une feuille UserForm est valide, c'est-à-dire qu'il correspond au type attendu. Si tel n'est pas le cas, il est probable que le programme génère une erreur. Celle-ci peut alors être évitée en convertissant le type de la variable.

Vérifier le type de données d'une variable

VBA intègre des fonctions vérifiant qu'une valeur correspond bien au type attendu. Elles sont présentées dans le tableau 6-3.

Tableau 6-3. Fonctions VBA vérifiant les types de données

Fonction	Description
`IsArray(MaVar)`	Renvoie `True` si *`MaVar`* est une variable de matrice ; `False` dans le cas contraire.
`IsDate(MaVar)`	Renvoie `True` si *`MaVar`* est une variable de date ; `False` dans le cas contraire.
`IsNumeric(MaVar)`	Renvoie `True` si *`MaVar`* est un nombre ; `False` dans le cas contraire.
`IsObject(MaVar)`	Renvoie `True` si *`MaVar`* est une variable objet ; `False` dans le cas contraire.
`IsMissing(MaVar)`	Renvoie `True` si l'argument optionnel *`MaVar`* est de type `Variant` et n'a pas été passé à la fonction ou à la procédure en cours.
`IsEmpty(MaVar)`	Renvoie `True` si *`MaVar`* n'a pas été initialisée, c'est-à-dire si aucune valeur ne lui a été affectée ; `False` dans le cas contraire. Valide uniquement pour les variables de type `Variant`.
`IsNull(MaVar)`	Renvoie `True` si *`MaVar`* contient la valeur `Null` ; `False` dans le cas contraire. Ne confondez pas une variable contenant une valeur `Null` et une variable qui n'a pas été initialisée et ne contient aucune valeur. Valide uniquement pour les variables de type `Variant`.
`IsError(MaVar)`	Renvoie `True` si *`MaVar`* stocke une valeur correspondant à l'un des codes d'erreur de VBA. `False` dans le cas contraire.

Vous pouvez également utiliser les fonctions `VarType` ou `TypeName` pour connaître le type d'une variable.

Utilisez `VarType` selon la syntaxe suivante :

```
MaVar = VarType(NomVar)
```

`MaVar` (de type `Integer`) reçoit pour valeur une constante Visual Basic indiquant le type de la variable *`NomVar`* (`vbInteger`, `vbDate`, etc.). `TypeName` s'utilise selon la même syntaxe, `MaVar =TypeName(NomVar)`, mais renvoie une chaîne de caractères représentant le type de la variable (voir tableau 6-4).

Tableau 6-4. Valeurs renvoyées par la fonction TypeName

Chaîne renvoyée	Variable
`Type objet`	

	Objet dont le type est *type_objet*
`Byte`	Octet
`Integer`	Entier
`Long`	Entier long
`Single`	Nombre à virgule flottante en simple précision
`Double`	Nombre à virgule flottante en double précision
`Currency`	Valeur monétaire
`Decimal`	Valeur décimale
`Date`	Valeur de date
`String`	Chaîne
`Boolean`	Valeur booléenne
`Error`	Valeur d'erreur
`Empty`	Non initialisée
`Null`	Aucune donnée valide
`Object`	Objet
`Unknown`	Objet dont le type est inconnu
`Nothing`	Variable objet qui ne fait pas référence à un objet

La fonction `EntrerUneDate` suivante utilise `InputBox` pour demander une date à l'utilisateur. `IsDate` est employée pour vérifier si la valeur entrée est bien valide. Si tel n'est pas le cas, l'utilisateur est invité à recommencer.

```
Function EntrerUneDate()
    Do
       EntrerUneDate = InputBox("Entrez une date", "Vérification du type de données")
    Loop Until IsDate(EntrerUneDate) = True
End Function
```

Info

La fonction `InputBox` et la structure de contrôle `Do…Loop` sont présentées au chapitre suivant.

Modifier le type d'une variable

VBA intègre des fonctions pour convertir une variable d'un type défini en une variable d'un autre type. Le tableau 6-5 les présente sommairement. Pour plus de précisions, reportez-vous à l'aide en ligne de VBA.

Tableau 6-5. Fonctions de conversion des types de données

Fonction	Description
`CBool(`*`MaVar`*`)`	Renvoie `True` si *`MaVar`* est une valeur numérique différente de 0 ; `False` si *`MaVar`* est égale à 0. Une erreur est générée si *`MaVar`* n'est pas une valeur numérique.

`CByte(MaVar)`	Convertit *`MaVar`* en une variable de type `Byte`.[1]
`CCur(MaVar)`	Convertit *`MaVar`* en une variable de type `Currency` (monétaire).[1]
`CDate(MaVar)`	Convertit *`MaVar`* en une variable de type `Date`.[1]
`CDbl(MaVar)`	Convertit *`MaVar`* en une variable de type `Double`.[1]
`CDec(MaVar)`	Convertit *`MaVar`* en une variable de type `Decimal`.[1]
`CInt(MaVar)`	Convertit *`MaVar`* en une variable de type `Integer`.[1]
`CLng(MaVar)`	Convertit *`MaVar`* en une variable de type `Long`.[1]
`CSng(MaVar)`	Convertit *`MaVar`* en une variable de type `Single`.[1]
`CVar(MaVar)`	Convertit *`MaVar`* en une variable de type `Variant`. *`MaVar`* doit être une valeur de type `Double` pour les nombres et de type `String` pour les chaînes.
`CStr(MaVar)`	Convertit *`MaVar`* en une variable de type `String`. Si *`MaVar`* est un booléen, `CStr` renvoie Vrai ou Faux. Si *`MaVar`* est une date, `CStr` la renvoie sous forme de chaîne. Si *`MaVar`* est un nombre, `CStr` renvoie cette valeur sous forme de chaîne.

1. `MaVar` doit être une valeur compatible avec le type de données vers lequel s'opère la conversion. Par exemple, si vous utilisez la fonction `Cbyte`, *`MaVar`* doit être une valeur numérique comprise entre 0 et 255. Sinon, une erreur « Type incompatible » est générée.

Notez que `MaVar` peut être une variable ou toute expression valide.

Portée et durée de vie des variables

Outre leurs type et valeur, les variables et les constantes sont caractérisées par leur *portée*. Ce terme désigne son accessibilité pour les procédures et les modules du projet. Variables et constantes peuvent être accessibles à une seule procédure, à l'ensemble des procédures d'un module, ou encore à l'ensemble des modules du projet en cours. Les variables sont aussi caractérisées par leur *durée de vie*, c'est-à-dire le temps pendant lequel elles conservent leur valeur : seulement pendant l'exécution d'une procédure, ou bien pendant toute l'exécution du programme.

Portée de niveau procédure

Une variable/constante est dite *de niveau procédure* lorsqu'elle n'est accessible qu'à la procédure dans laquelle elle est déclarée.

Portée de niveau module privée

Une variable/constante est dite *de niveau module privée* lorsqu'elle est

accessible à l'ensemble des procédures du module dans lequel elle est déclarée. Elle doit pour cela être déclarée dans la section Déclarations du module, c'est-à-dire à l'extérieur de toute procédure.

Par défaut, les variables/constantes déclarées ainsi ont une portée *privée*, c'est-à-dire qu'elles ne sont accessibles qu'aux procédures du module. Vous pouvez cependant substituer le mot-clé `Private` à `Dim` pour améliorer la lisibilité de votre code. La déclaration de la variable se présente alors ainsi :

```
Private NomVariable As Type
```

Dans l'exemple suivant, la constante `Pi` est déclarée de niveau module privée et est accessible à toutes les procédures du module.

```
Option Explicit
Private Pi As Single
Pi = 3.14


Sub Procédure-1 ()
[…]  'Instructions
End Sub


[…]  'Autres procédures du module


Sub Procédure-n ()
[…]  'Instructions
End Sub
```

Portée de niveau module publique

Une variable/constante est dite *de niveau module publique* lorsqu'elle est accessible à toutes les procédures du projet, quel que soit le module de stockage. Elle doit pour cela être déclarée dans la section Déclarations du module, à l'aide de l'instruction `Public`, selon la syntaxe suivante :

```
Public NomVariable As Type
```

Conseil

Lorsque vous utilisez une valeur définie de façon récurrente dans un projet (une TVA, par exemple), affectez-lui une constante de niveau module publique et utilisez cette constante plutôt que la valeur elle-même dans les procédures. Si cette valeur est modifiée, il vous suffira de redéfinir l'instruction d'affectation de la constante pour mettre à jour la totalité du projet.

Variables statiques

Une variable conserve une valeur, modifiable, tant que le programme

s'exécute dans son champ de portée. Lorsque l'exécution du programme sort de la portée de la variable, celle-ci est réinitialisée et perd sa valeur. Autrement dit, une variable de niveau procédure conserve sa valeur tant que la procédure dans laquelle elle est déclarée est en cours d'exécution – même lorsqu'elle appelle d'autres procédures. Lorsque celle-ci se termine, la variable est réinitialisée. Une variable de niveau module conserve sa valeur jusqu'à ce que le programme prenne fin.

Pour qu'une variable de niveau procédure conserve sa valeur entre différents appels, substituez le mot-clé `Static` à `Dim` dans l'instruction de déclaration :

```
Static NomVariable As Type
```

Astuce

Pour déclarer statiques toutes les variables d'une procédure `Sub` ou `Function`, placez le mot-clé `Static` devant l'instruction de déclaration de la procédure.

Traitement entre applications à l'aide de variables objets

Une variable objet peut se voir affecter un objet appartenant à une autre application que l'hôte du projet. Un programme VBA Excel sait ainsi exploiter des objets de Word, Access ou toute autre application supportant *Automation*. Il suffit pour cela d'affecter à une variable l'objet auquel vous souhaitez accéder, à l'aide des fonctions `GetObject` et/ou `CreateObject`.

Définition

Automation, ou OLE Automation, est une fonction du modèle d'objets Composant (COM, Component Object Model). Il s'agit d'un standard qu'utilisent les applications pour exposer leurs objets, méthodes et propriétés aux outils de développement. Les applications Office supportent Automation. Un classeur Excel peut ainsi exposer une feuille de calcul, un graphique, une cellule ou une plage de cellules, etc. Un fichier Word exposera une page, un paragraphe, un mot, ou tout autre objet de son modèle. Visual Basic pour Applications sait accéder à ces objets, interroger ou redéfinir leurs propriétés, en exécuter les méthodes, etc.

Pour qu'un projet accède à la bibliothèque d'une autre application, celle-ci doit être référencée dans le projet : choisissez la commande Références du menu Outils. Dans la boîte de dialogue qui s'affiche, cochez les cases des bibliothèques qui vous intéressent, puis cliquez sur OK.

Pour réaliser l'exemple suivant, référencez la bibliothèque d'objets Microsoft Word Object Library à partir d'un projet Excel. Puis créez un nouveau

document Word et enregistrez-le sur le Bureau de Windows, sous le nom MonDoc.docx.

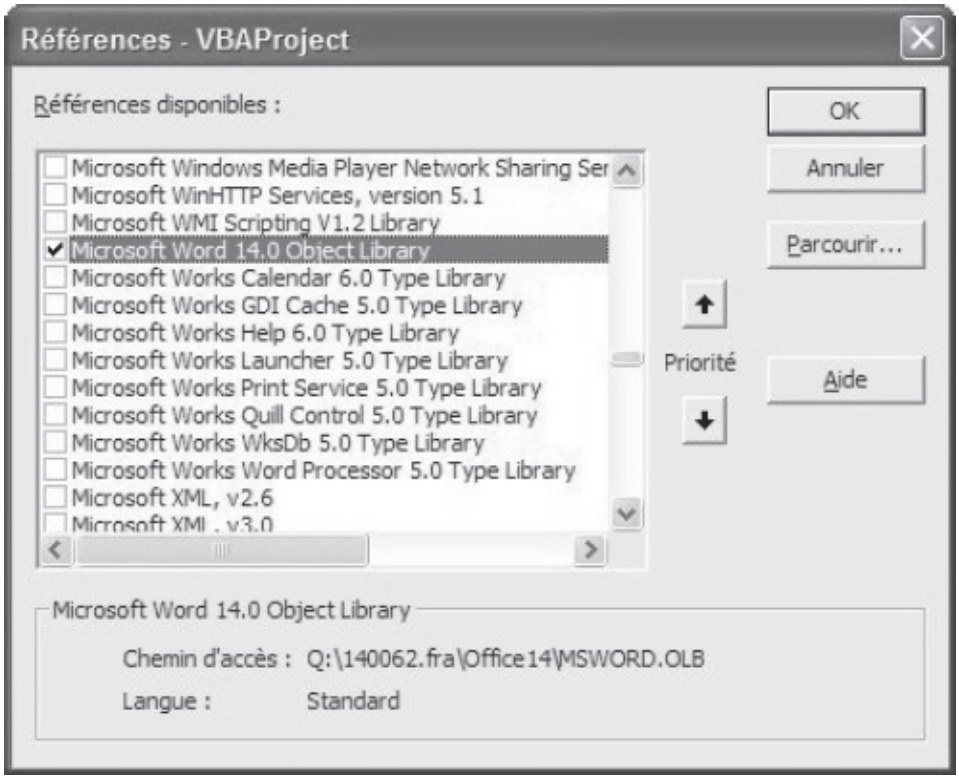

Références - VBAProject
Références disponibles :
OK
Annuler
Parcourir...
Aide
Priorité
Microsoft Windows Media Player Network Sharing Ser
Microsoft WinHTTP Services, version 5.1
Microsoft WMI Scripting V1.2 Library
Microsoft Word 14.0 Object Library
Microsoft Works Calendar 6.0 Type Library
Microsoft Works GDI Cache 5.0 Type Library
Microsoft Works Help 6.0 Type Library
Microsoft Works Launcher 5.0 Type Library
Microsoft Works Print Service 5.0 Type Library
Microsoft Works Quill Control 5.0 Type Library
Microsoft Works WksDb 5.0 Type Library
Microsoft Works Word Processor 5.0 Type Library
Microsoft XML, v2.6
Microsoft Word 14.0 Object Library
Chemin d'accès : Q:\140062.fra\Office14\MSWORD.OLB
Langue : Standard

Figure 6-14 – *Activez la bibliothèque d'objets de l'application que vous souhaitez manipuler à partir de la boîte de dialogue Références.*

Placez ensuite le code suivant dans un module Excel :

```
 1: Sub InsereTableauDansFichierWord()
 2:   Dim MonDoc As Object
 3:   On Error Resume Next
 4:     Set MonDoc = GetObject(, "Word.Application")
 5:     If Err.Number<>0 Then Err.Clear
 6:   Set MonDoc = GetObject("C:\Users\Nom_utilisateur\Desktop\MonDocx.doc")
 7:   Dim MaPosition As Word.Range
 8:   Set MaPosition = MonDoc.Range(0,0)
 9:   MonDoc.Tables.Add Range:=MaPosition, NumRows:=3, NumColumns:=4
10:   MonDoc.Save
11:   Set MonDoc = Nothing
12:   Word.Application.Quit
13: End Sub
```

Attention

Veillez à personnaliser le chemin précisé pour la fonction `GetObject` à la ligne 6, sinon cette macro ne fonctionnera pas.

Exécutez la procédure, puis ouvrez le fichier Word. Un tableau de quatre

colonnes sur trois lignes a été placé en début de document.

La variable objet `MonDoc` est déclarée ligne 2. Elle se voit ensuite affecter l'objet `Word.Application`, représentant l'application Word, à l'aide de l'instruction `Set` et de la fonction `GetObject` (ligne 4). Si Word n'est pas ouvert, une erreur est générée. Un détecteur d'erreurs est donc placé en ligne 3, de façon à ignorer l'erreur et à passer à l'instruction suivante. Si, effectivement, une erreur survient (`If Err.Number<>0`), la propriété `Number` de l'objet `Err` est redéfinie à 0 (ligne 5). Cette éventuelle erreur étant gérée, `MonDoc` peut recevoir `MonDoc.docx` (ligne 6).

La variable `MaPosition` de type `Word.Range` est déclarée ligne 7 – l'objet `Range` de Word représente une position de curseur dans un document. Ligne 8, la position représentant le début du document lui est affectée. Un tableau est inséré à cette position ligne 9. Le document est ensuite sauvegardé. Ligne 11, la variable `MonDoc` est libérée. Enfin, l'instruction de la ligne 12 quitte Word. En effet, lorsque vous faites appel à une variable objet d'une autre application, son moteur est lancé. N'omettez donc pas d'employer la méthode `Quit`, afin de libérer les ressources occupées.

Info

On distingue, dans l'accès aux objets d'autres applications à l'aide d'Automation, la liaison tardive de la liaison précoce. La liaison est dite tardive lorsqu'une variable de type `Object` ou `Variant` est déclarée (`Dim MaVar As Object`). La variable est ensuite initialisée et affectée à un objet de l'application étrangère à l'aide de la fonction `GetObject`. On parle de liaison précoce lorsque la variable est déclarée en choisissant un type identifiant l'application dont on souhaite exploiter les objets (`Dim MaVar as Word.Application`). Utilisez de préférence une liaison précoce dans vos programmes. Les performances en seront améliorées et Visual Basic vérifiera la syntaxe spécifique aux objets de l'application étrangère lors de l'écriture de votre code.

Notez qu'un programme Excel peut exécuter une macro stockée dans une autre application hôte. Dans l'exemple suivant, la macro MacroWord est exécutée sur un nouveau document à partir d'Excel. Pour réaliser cet exemple, commencez par créer MacroWord :

1. Lancez Word. Définissez le niveau de sécurité de façon à autoriser l'exécution des macros (commande Sécurité des macros de l'onglet Développeur).

Info

Le modèle Normal.dotm de Word est l'équivalent du classeur de macros personnel d'Excel : les macros stockées dans ce modèle sont accessibles à tous les documents Word.

2. Activez l'onglet Développeur, puis cliquez sur la commande Macros. Avec une version de Word antérieure à 2007, choisissez Outils > Macro > Macros.
3. Dans la zone Nom de la macro, saisissez MacroWord et, dans la liste déroulante Macros disponibles dans, sélectionnez Normal.dot (modèle global).
4. Cliquez sur le bouton Créer. Visual Basic Editor s'ouvre sur la fenêtre Code de MacroWord.
5. Complétez le code de la macro de la façon suivante :

```
Sub MacroWord()
    MsgBox "Cette boîte de dialogue est affichée par la macro MacroWord", _
    vbOKOnly + vbInformation, "Exécution d'une macro Word à partir d'un programme Excel"
End Sub
```

6. Enregistrez, puis fermez Word.

Retournez à Visual Basic Editor pour Excel et créez la procédure suivante :

```
1: Sub ExecuterMacroWord()
2:    Dim MonWord As Object
3:    Set MonWord = CreateObject("Word.Application")
4:    MonWord.Visible = True
5:    MonWord.Documents.Add
6:    MonWord.Run "MacroWord"
7:    Set MonWord = Nothing
8: End Sub
```

Exécutez la procédure `ExecuterMacroWord`. La boîte de dialogue représentée à la figure 6-15 s'affiche.

Ligne 2, la variable `MonWord` est créée et reçoit l'objet `Word.Application` à la ligne suivante. Ligne 4, la propriété `Visible` de `MonWord` est définie à `True` afin d'afficher Word à l'écran. Un document est ensuite créé. Ligne 6, on applique la méthode `Run` à `MonWord` afin d'exécuter MacroWord. La boîte de dialogue représentée à la figure 6-15 s'affiche alors. Ligne 7, la variable objet est libérée et le programme prend fin.

Figure 6-15 – *Une macro Excel peut contrôler l'exécution de macros dans*

d’autres applications hôtes.

Info

Le programme complet présenté au chapitres 17 fournit un bon exemple de traitement entre applications, puisqu’il propose notamment d’éditer des documents Word complexes à partir de données traitées dans le programme Excel.

7

Contrôler les programmes VBA

Visual Basic intègre des instructions orientant le comportement d'une macro : les *structures de contrôle* – on parle du flux de contrôle d'un programme. La connaissance et la maîtrise de ces structures constituent un préalable indispensable à la création de programmes VBA souples et puissants, se comportant différemment selon l'état du document et de l'application au cours de son exécution, ou suivant les informations fournies par l'utilisateur.

Ce chapitre aborde une à une les structures de contrôle de Visual Basic. Leur combinaison vous fera gagner un temps précieux dans vos tâches les plus communes comme les plus complexes. L'instruction `GoTo` et les fonctions `MsgBox` et `InputBox`, ainsi que la collection `Dialogs`, sont également traitées dans ce chapitre. Il ne s'agit pas de structures de contrôle, mais elles servent aussi à orienter le comportement des programmes VBA et à interagir avec l'utilisateur.

Répéter une série d'instructions : les boucles

Une *boucle* est un ensemble d'instructions se répétant en série un certain nombre de fois, ce nombre étant déterminé dans le code ou indéterminé, en fonction du contexte au moment de l'exécution du programme.

- **`Do…Loop`** et **`While…Wend`** généralisent une série d'instructions particulières à l'ensemble d'un document ; dans ce cas, ce sont l'état du document et celui de l'application qui déterminent le nombre de boucles réalisées.
- **`For…Next`** répète une série d'instructions sur un document un nombre de fois déterminé par l'utilisateur.
- **`For Each…Next`** exécute une série d'instructions sur tous les objets d'une collection.

La boucle While…Wend

La structure de contrôle `While…Wend` répète une série d'instructions tant qu'une condition spécifiée est remplie. C'est l'une des structures les plus utilisées pour automatiser les tâches répétitives.

Sa syntaxe est la suivante :

```
While Condition
   Série d'instructions
Wend
```

où *`Condition`* est une expression comparant deux valeurs à l'aide d'un opérateur relationnel. Lorsque la condition spécifiée après `While` est réalisée, le programme exécute la *`Série d'instructions`*. Lorsque l'instruction `Wend` est atteinte, le programme retourne à `While` et teste à nouveau la condition. Si elle est réalisée, la *`Série d'instructions`* s'exécute à nouveau, etc. Dans le cas contraire, le bloc placé entre `While` et `Wend` est ignoré ; et l'exécution du programme se poursuit alors avec l'instruction située immédiatement après `Wend`.

Pour poser une condition, on conjugue généralement une expression avec un *opérateur relationnel*, ou *opérateur de comparaison*, et une valeur. L'opérateur relationnel établit un rapport entre le résultat renvoyé par l'expression et la valeur. Si ce rapport est vérifié, la condition est respectée.

Le tableau 7-1 présente les opérateurs relationnels de Visual Basic.

Tableau 7-1. Les opérateurs relationnels de Visual Basic

Opérateur relationnel	Signification
`=`	Égal à
`>`	Supérieur à
`<`	Inférieur à
`<>`	Différent de
`>=`	Supérieur ou égal à
`<=`	Inférieur ou égal à
`Like`	Identique à (pour comparer des chaînes de caractères)
`Is`	Égal à (pour comparer des variables objets)

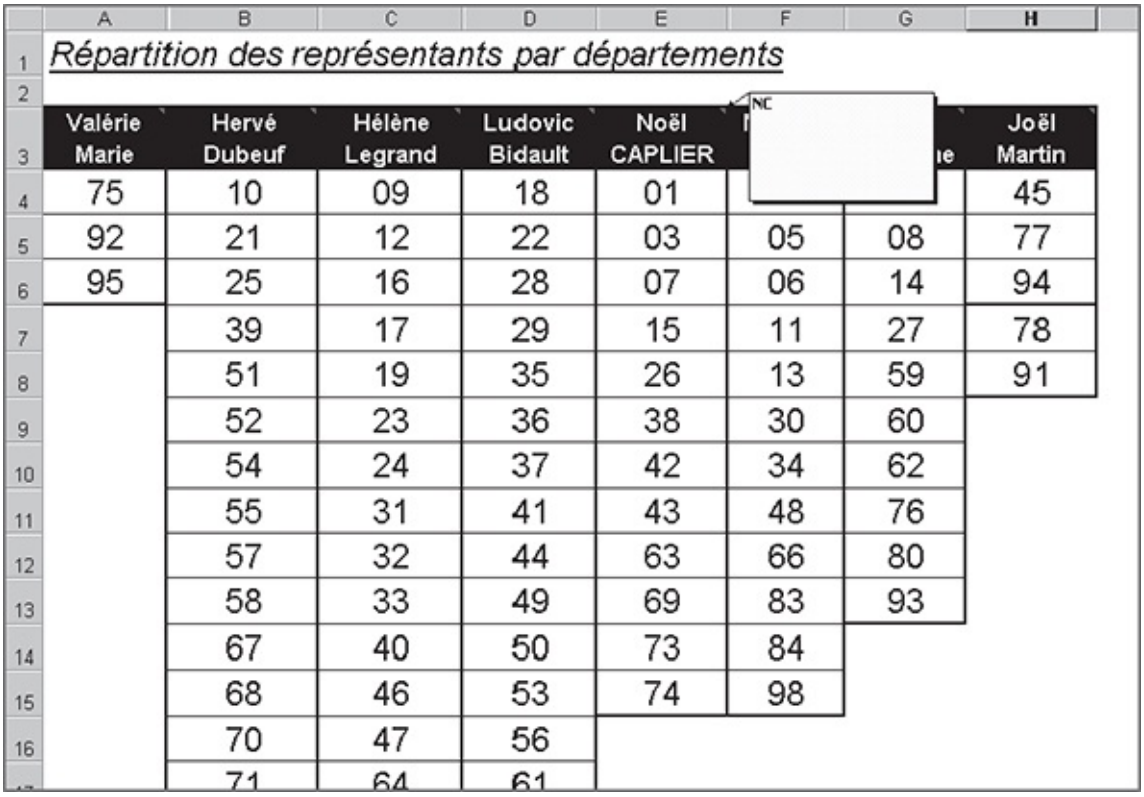
Conseil

Deux chaînes de caractères peuvent être comparées à l'aide des opérateurs relationnels =, <, >, etc. L'opération s'effectue alors entre les codes ANSI attachés aux caractères. Si vous devez effectuer des comparaisons précises, préférez l'opérateur `Like`. Celui-ci permet en effet de prendre ou non en compte la casse et d'utiliser des caractères génériques. Consultez l'aide en ligne pour plus de précisions.

La technique la plus courante pour enregistrer des instructions en boucle consiste à exécuter la *`série d'instructions`* après avoir activé l'Enregistreur de

macro, puis à ouvrir la fenêtre Code de la macro et à y insérer la structure `While…Wend`.

Nous utiliserons une structure `While…Wend` pour automatiser la saisie d'informations dans une feuille de calcul. Considérez le classeur Representants par departements représenté à la figure 7-1. Dans la feuille de calcul active (libellée Representants), les cellules de la ligne 3 contiennent chacune le nom d'un représentant et, en commentaire, ses initiales. Les colonnes correspondantes contiennent les numéros des départements dont chaque représentant a la charge.



Répartition des représentants par départements

Valérie Marie	Hervé Dubeuf	Hélène Legrand	Ludovic Bidault	Noël CAPLIER			Joël Martin
75	10	09	18	01			45
92	21	12	22	03	05	08	77
95	25	16	28	07	06	14	94
	39	17	29	15	11	27	78
	51	19	35	26	13	59	91
	52	23	36	38	30	60	
	54	24	37	42	34	62	
	55	31	41	43	48	76	
	57	32	44	63	66	80	
	58	33	49	69	83	93	
	67	40	50	73	84		
	68	46	53	74	98		
	70	47	56				
	71	64	61				

Figure 7-1 – *La répartition des représentants par départements.*

Le classeur Representants par clients, illustré à la figure 7-2, contient la liste des clients de la société (colonne A), la ville de chacun (colonne B) et son numéro (colonne D). Les deux premiers chiffres de ce dernier correspondent au département d'origine du client. La colonne C contiendra les initiales du représentant en charge du client. Nous profiterons de ce que ces deux classeurs ont en commun le numéro du département pour automatiser la mise à jour de la colonne C.

	A	B	C	D
1	Répartition des clients par représentant			
2				
3	Client	Ville	Représentant	N° Client
4	La Loterie	Joigny		8900002
5	Le Soldeur	Puteaux		9200003
6	Centre Soldes	Antibes		0600006
7	Prix Gros	Nantes		4400007
8	Le Grossiste	Saint Nazaire		4400015
9	La bonne affaire	La Rochelle		1700006
10	Le bon choix	Paris		7503009
11	Valable	Gannat		0300003
12	Facile	Marseille		1300046
13	C'est ici	Chalon		7100001
14	Prix écrasés	Colmar		6800014
15	Unique prix	Lyon		6900009

Figure 7-2 – *La répartition des représentants par clients avant mise à jour de la colonne C.*

La macro suivante extrait les deux premiers chiffres du numéro de client. Elle recherche ensuite cette valeur dans le classeur Representants par departement, de façon à identifier le représentant en charge du client, dont les initiales sont alors insérées dans la cellule correspondante de la colonne C. La structure `While…Wend` répète cette procédure en boucle. Chaque fois que les initiales d'un représentant ont été insérées, la cellule Numéro de client suivante est activée. La procédure s'exécute TANT QUE la cellule sélectionnée contient une valeur.

```
 1: Sub InsererInitialesRepresentants()
 2:   Dim ClasseurRepresentants As Workbook
 3:   Dim NumDepartement As String
 4:   Dim Colonne As Variant
 5:   Dim Initiales
 6:   Set ClasseurRepresentants = _
        GetObject("C:\Users\Nom_utilisateur\Desktop\Representants par
        ➥ departements.xlsx")
 7:   Range("D4").Select
 8:   While ActiveCell.Value<>""
 9:      NumDepartement = Left(ActiveCell.Value,2)
10:      Colonne = ClasseurRepresentants.Sheets(1).Range("A4:I50").Find(What:=
         ➥ NumDepartement, LookIn:=xlFormulas, LookAt:=xlWhole).Address
11:      Colonne = Range(Colonne).Column
12:      Colonne = CInt(Colonne)
13:      Initiales = ClasseurRepresentants.Sheets(1).Cells(3,Colonne).Comment.Text
14:      ActiveCell.Offset(0,-1).Range("A1").Select
15:      ActiveCell.FormulaR1C1 = Initiales
16:      ActiveCell.Offset(1,1).Range("A1").Select
17:   Wend
18:   Set ClasseurRepresentants = Nothing
```

```
19:   Workbooks("Representants par departements.xlsx").Close
20: End Sub
```

Lignes 2 à 6 : les variables qui seront exploitées par le programme sont déclarées, et à `ClasseurReprésentant` est affecté le classeur Representants par departements.xlsx, situé sur le Bureau. Ligne 7, la cellule D4 est sélectionnée.

La boucle `While…Wend` des lignes 8 à 17 s'exécute tant que la cellule sélectionnée contient des informations. Ligne 9, la fonction `Left` affecte à la variable `NumDepartement` les deux caractères de gauche (correspondant au numéro de département) de la valeur de la cellule active. Cette valeur est ensuite recherchée dans le classeur des représentants par départements (ligne 10). L'objet `Range` renvoyé par la méthode `Find` est affecté à `Colonne` – notez que cette variable a été déclarée de type `Variant` de sorte qu'elle puisse recevoir des valeurs de différents types. Ligne 11, `Colonne` reçoit la valeur correspondant au numéro de la colonne de la cellule trouvée. Ligne 12, la fonction `CInt` convertit la valeur de `Colonne` en un `Integer`, ce qui la rend utilisable comme argument de la propriété `Cells`. Ligne 13, la variable `Initiales` reçoit pour valeur les initiales du représentant en charge du département : on lui affecte pour ce faire le texte de commentaire de la cellule située dans la même colonne que l'objet `Range` renvoyé par la fonction `Find`, mais sur la ligne 3 – la ligne des noms de représentants.

Lignes 14 et 16, un déplacement par référence relative aux cellules est effectué. Tout d'abord, la case située à gauche de la cellule active est sélectionnée et reçoit la valeur de la variable `Initiales` (ligne 15). Un déplacement d'une cellule vers la droite puis d'une vers le bas est ensuite effectué. La cellule active est alors la suivante dans la colonne D. Le mot-clé `Wend` renvoie l'exécution du programme à l'instruction `While` correspondante. Celle-ci vérifie que la cellule active contient des données. Si tel est le cas, le processus recommence. Lorsque la condition n'est plus vérifiée, les instructions situées entre `While` et `Wend` sont ignorées et le programme se termine avec les instructions des lignes 18 et 19. Les ressources système occupées par la variable objet `ClasseurRepresentants` sont libérées et le classeur est fermé.

Conseil

Pour sécuriser définitivement cette macro, commencez par lui faire activer la feuille devant recevoir les informations. Elle devra logiquement être stockée dans ClasseurRepresentants.xlsx, puisqu'elle ne servira qu'à ce classeur. Ainsi, elle ne s'exécutera que si le classeur est ouvert.

	A	B	C	D	E
1	Répartition des clients par représentant				
2					
3	Client	Ville	Représentant	N° Client	Solde
4	La Loterie	Joigny	HD	8900002	
5	Le Soldeur	Puteaux	VM	9200003	
6	Centre Soldes	Antibes	MF	0600006	
7	Prix Gros	Nantes	LB	4400007	
8	Le Grossiste	Saint Nazaire	LB	4400015	
9	La bonne affaire	La Rochelle	HL	1700006	
10	Le bon choix	Paris	VM	7503009	
11	Valable	Gannat	NC	0300003	
12	Facile	Marseille	MF	1300046	
13	C'est ici	Chalon	HD	7100001	
14	Prix écrasés	Colmar	HD	6800014	
15	Unique prix	Lyon	NC	6900009	
16	Aux bas prix	Frejus	MF	8300001	
17	Exceptionnel	Colmar	HD	6800014	
18	Venez tous	Chalon	HD	7100001	
19					

Figure 7-3 – *La macro a complété les informations de la colonne C.*

La boucle Do…Loop

La structure de contrôle `Do…Loop` est semblable à `While…Wend` ; mais elle offre plus de souplesse, car elle peut se décliner sur quatre modes différents :

- **`Do While…Loop`**. Tant que la condition est respectée, la boucle s'exécute.

```
Do While Condition
  Série d'instructions
Loop
```

- **`Do Until…Loop`**. Jusqu'à ce que la condition soit réalisée, la boucle s'exécute.

```
Do Until Condition
  Série d'instructions
Loop
```

- **`Do…Loop While`**. La boucle s'exécute, puis se répète si la condition est respectée.

```
Do
  Série d'instructions
Loop While Condition
```

- **`Do…Loop Until`**. La boucle s'exécute, puis se répète jusqu'à ce que la condition soit respectée.

```
Do
  Série d'instructions
Loop Until Condition
```

Le programme suivant utilise une boucle `Do While…Loop` pour supprimer les doublons dans un classeur Excel (figures 7-4 et 7-5). On estime, dans cette première version, qu'il existe un doublon lorsque deux cellules de la colonne A contiennent les mêmes données. Le programme commence par trier ces dernières. Le contenu de chaque cellule de la colonne A est ensuite comparé à celui de la cellule suivante. S'ils sont identiques, la ligne de la cellule courante est supprimée.

```
 1: Sub SuppressionDoublons()
 2:    Dim CelluleCourante As Range
 3:    Dim CelluleSuivante As Range
 4:    Set CelluleCourante = ActiveSheet.Range("A1")
 5:
 6:    'Tri des données sur la cellule A1
 7:    ActiveSheet.Range("A1").Sort key1:=Range("A1"), _
 8:       Order1:=xlAscending, Header:= xlGuess, OrderCustom:=1, _
 9:       MatchCase:=False, Orientation:=xlTopToBottom
10:    'Boucle
11:    Do While IsEmpty(CelluleCourante) = False
12:       Set CelluleSuivante = CelluleCourante.Offset(1,0)
13:       If CelluleSuivante.Value = CelluleCourante.Value Then
14:          CelluleCourante.EntireRow.Delete
15:       End If
16:       Set CelluleCourante = CelluleSuivante
17:    Loop
18: End Sub
```

Lignes 2 et 3, les variables objets `CelluleCourante` et `CelluleSuivante` sont déclarées. `CelluleCourante` reçoit ensuite un objet `Range` correspondant à la cellule A1 de la feuille active. L'instruction des lignes 7 à 9 trie les données. On applique pour cela la méthode `Sort`. Les arguments `Key1` et `Order1` définissent respectivement le premier critère de tri et l'ordre du tri. `Header` reçoit ici la constante `xlGuess` (Excel définit s'il y a ou non une ligne de titre et, dans l'affirmative, de quelle ligne il s'agit). `OrderCustom` reçoit la valeur 1 et le tri est donc « Normal ». Enfin, `MatchCase` et `Orientation` correspondent au respect de la casse lors du tri et à son orientation (ici de haut en bas).

Lignes 11 à 17, une boucle `Do While…Loop` est utilisée pour tester toutes les cellules. `CelluleCourante` est testée, puis reçoit la valeur stockée dans `CelluleSuivante`. La boucle s'exécute tant que `CelluleCourante` n'est pas vide [`IsEmpty(CelluleCourante) = False`].

Ligne 12, la propriété `Offset` est utilisée pour attribuer à `CelluleSuivante` la case en-dessous dans la même colonne. Lignes 13 à 15, une instruction conditionnelle supprime la ligne de `CelluleCourante` (`CelluleCourante.EntireRow`) s'il y a un doublon avec la cellule suivante. `CelluleCourante` reçoit ensuite la cellule

stockée dans `CelluleSuivante` (ligne 16).

Ligne 17, l'instruction `Loop` renvoie le programme à l'instruction `While` correspondante. La condition est de nouveau évaluée, et le corps de la boucle s'exécute si elle est vérifiée. Lorsqu'elle n'est plus vérifiée, le programme se poursuit avec l'instruction située immédiatement sous `Loop`. En l'occurrence, il prend fin.

	A	B	C	D	E	F	G	H
1	aaaa	1						
2	bbbb	2						
3	cccc	3						
4	aaaa	4						
5	bb	5						
6	cc	6						
7	aa	7						
8	bbbb	8						
9	cccc	9						
10								
11								
12								
13								

Figure 7-4 – *La feuille avant passage de la macro.*

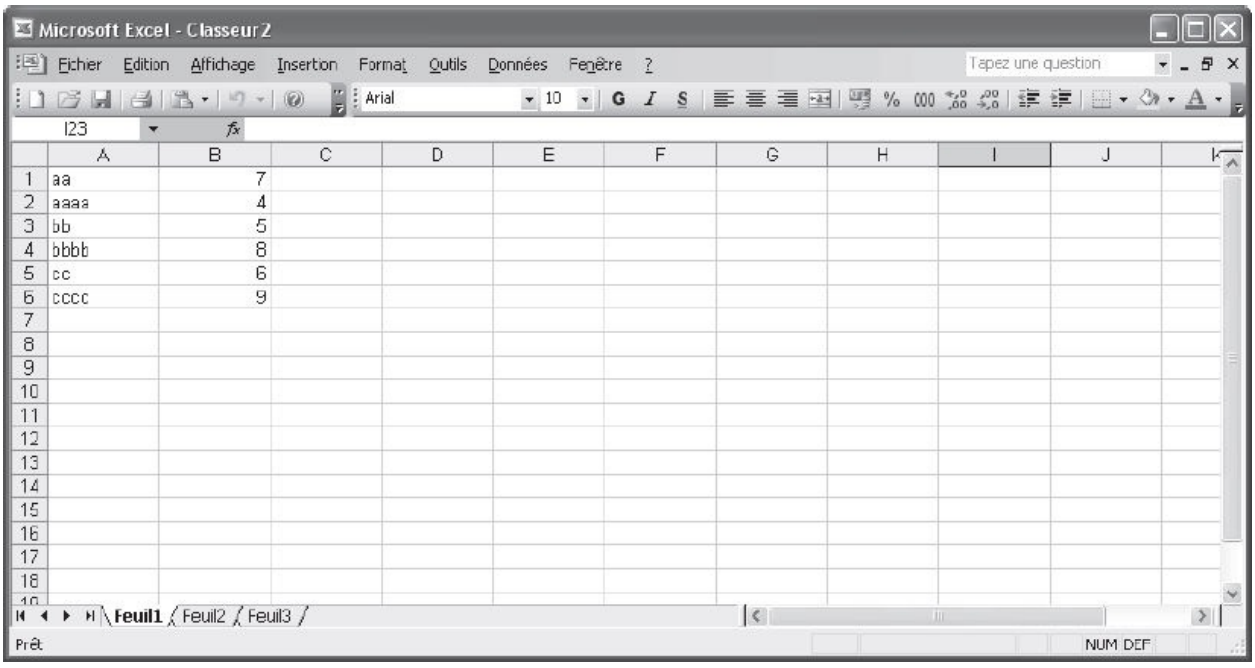


Figure 7-5 – *La macro a supprimé les doublons.*

Le programme fonctionne correctement, mais ne prend en compte que le contenu des cellules de la colonne A pour déterminer les doublons. La

procédure suivante supprime une ligne uniquement si les données sont également identiques dans les colonnes B, C et D.

Attention

Veillez à ajouter des données dans les colonnes C et D avant d'exécuter la nouvelle version de la macro SuppressionDoublons.

```
 1: Sub SuppressionDoublons()
 2:    Dim Cellulecourante As Range
 3:    Dim Cellulesuivante As Range
 4:    Set Cellulecourante = ActiveSheet.Range("A1")
 5:
 6:    'Tri des données sur la cellule A1
 7:    ActiveSheet.Range("A1").Sort Key1:=Range("A1"), Order1:=xlAscending,
       ➥ Key2:=Range("B1"), _
 8:       Order2:=xlAscending, Key3:=Range("C1"), Order3:=xlAscending,
       ➥ Header:=xlGuess, _
 9:       OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom
10:    'Boucle et test des cellules
11:    Do While IsEmpty(Cellulecourante) = False
12:       Set Cellulesuivante = Cellulecourante.Offset(1,0)
13:       If Cellulesuivante.Value = Cellulecourante.Value Then
14:          If LignesIdentiques(Cellulecourante, Cellulesuivante) = True Then
15:             Cellulecourante.EntireRow.Delete
16:          End If
17:       End If
18:       Set Cellulecourante = Cellulesuivante
19:    Loop
20: End Sub
21:
22: Function LignesIdentiques(CellCourante As Range, CellSuivante As Range)
     ➥ As Boolean
23:   If CellCourante.Offset(0,1).Value <> CellSuivante.Offset(0,1).Value Then
24:      LignesIdentiques = False
25:   ElseIf CellCourante.Offset(0,2).Value <> CellSuivante.Offset(0,2).Value Then
26:      LignesIdentiques = False
27:   ElseIf CellCourante.Offset(0,3).Value <> CellSuivante.Offset(0,3).Value Then
28:      LignesIdentiques = False
29:   Else
30:      LignesIdentiques = True
31:   End If
32: End Function
```

Ligne 15, l'instruction conditionnelle définissant si la ligne est supprimée appelle la fonction `LignesIdentiques` en lui passant les arguments `CelluleCourante` et `CelluleSuivante`. Cette fonction est déclarée comme recevant deux arguments de type `Range` et renvoyant un booléen (ligne 22).

Lignes 23 à 31, une structure conditionnelle détermine la valeur renvoyée par la fonction. Le contenu des cellules – décalées d'une, de deux, puis de trois cases à droite de `CelluleCourante` – est successivement comparé au contenu des cellules décalées de la même façon par rapport à `CelluleSuivante` [`Offset(0,1)`, `Offset(0,2)` et `Offset(0,3)`]. Si ce contenu diffère, la valeur `False` est affectée à la fonction (lignes 24, 26 et 28). Dans le cas contraire, la fonction renvoie `True`

(ligne 30). La procédure appelante reprend alors la main, et l'instruction de la ligne 15 est exécutée si la fonction a renvoyé `True`. Dans le cas contraire, la condition n'est pas vérifiée et la cellule suivante est testée.

Rappel

Pour interrompre une macro qui ne fonctionne pas correctement (qui exécute une boucle sans fin, par exemple), appuyez sur Ctrl+Pause, puis voyez le chapitres 10.

La boucle For…Next

La structure de contrôle `For…Next` répète une série d'instructions un nombre de fois déterminé dans le code, en utilisant un compteur et selon la syntaxe suivante :

```
For compteur = x To y Step Pas
   série d'instructions
Next compteur
```

La macro exécute en boucle la série d'instructions spécifiée entre `For` et `Next`, en incrémentant la variable *`compteur`* de la valeur de *`Pas`* à chaque passage de la boucle. Si l'argument `Step` est omis, le compteur est incrémenté de 1. Tant que la valeur de *`compteur`* est inférieure à *`y`*, la boucle se répète ; lorsque la condition n'est plus vérifiée, la procédure se poursuit avec les instructions situées derrière l'instruction `Next`.

La procédure suivante applique un ombrage de cellules à une ligne sur deux d'une feuille de calcul Excel, afin d'obtenir une mise en forme semblable à celle représentée à la figure 7-6.

```
 1: Sub FormaterClasseur()
 2:   Dim compteur As Integer
 3:   Dim MaLigne As Variant
 4:   Cells.Interior.ColorIndex = 2
 5:   MaLigne = Range("A1").End(xlDown).Address
 6:   MaLigne = Range(MaLigne).Row
 7:     If Not MaLigne/2 = Int(MaLigne/2) Then
 8:       MaLigne = MaLigne + 1
 9:     End If
10:     For compteur = 2 To MaLigne Step 2
11:       Range(compteur & ":" & compteur).Select
12:       Selection.Interior.ColorIndex = 15
13:     Next compteur
14: End Sub
```

Deux variables sont tout d'abord déclarées. La propriété `ColorIndex` de l'objet `Interior` de tous les objets de la collection `Cells` est ensuite définie à 2 (ligne 4) – ce qui revient à appliquer la couleur de fond blanche à l'ensemble des cellules de la feuille active.

Les instructions des lignes 4 à 9 servent à déterminer jusqu'à quelle ligne le formatage doit s'effectuer. L'adresse de la dernière cellule non vide sous la cellule A1 est affectée à la variable `MaLigne` (ligne 5), qui reçoit ensuite pour valeur le numéro de ligne de cette cellule (ligne 6). Une instruction conditionnelle `If…End If` est utilisée pour vérifier que `MaLigne` est une valeur paire (lignes 7 à 9). Si tel n'est pas le cas (si `MaLigne` divisée par 2 n'est pas un nombre entier), `MaLigne` est incrémentée de 1.

La boucle `For…Next` peut maintenant être exécutée. Le compteur de la boucle commence à 2 et est incrémenté de 2 à chaque passage de la boucle, jusqu'à atteindre la valeur `MaLigne` (ligne 10). À chaque passage de la boucle, la ligne correspondant à la valeur de la variable compteur est sélectionnée (ligne 11), et l'ombrage de cellule correspondant à la valeur 15 de la propriété `ColorIndex` lui est appliqué (ligne 12).

	A	B	C	D	E	F	G	H
1	**Vendeur**	**Région**	**Ville**	**Chiffre d'A.**				
2	Mary	Nord	Lille					
3	du Château	Ile de France	Paris					
4	Bi Do	Ouest	Quimper					
5	Des neury	Sud Ouest	Biarritz					
6	Gas Quai	Sud Est	Marseille					
7	Ussu Nez	Nord Ouest	Cherbourg					
8	Fernand	Centre	Saint Etienne					
9	Bagousse	Est	Châlon s/Saône					
10								
11								
12								

Figure 7-6 – *Une mise en forme automatisée.*

Le programme suivant constitue une nouvelle version de la procédure `SuppressionDoublons`, dans laquelle la fonction `LignesIdentiques` a été améliorée. La fonction utilise maintenant une boucle `For…Next` pour définir le déplacement (`Offset`) effectué lors des comparaisons. Par ailleurs, le nombre de cellules à comparer afin de définir si une ligne constitue un doublon est défini lors de l'appel de la fonction.

```
1: Sub SuppressionDoublons()
2:    Dim Cellulecourante As Range
3:    Dim Cellulesuivante As Range
4:    Set Cellulecourante = ActiveSheet.Range("A1")
```

```
 5:
 6:    'Tri des données sur la cellule A1
 7:     ActiveSheet.Range("A1").Sort Key1:=Range("A1"), Order1:=xlAscending,
          ➥ Key2:=Range("B1"), _
 8:        Order2:=xlAscending, Key3:=Range("C1"), Order3:=xlAscending,
          ➥ Header:=xlGuess, _
 9:        OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom
10:    'Boucle et test des cellules
11:    Do While IsEmpty(Cellulecourante) = False
12:       Set Cellulesuivante = Cellulecourante.Offset(1,0)
13:       If Cellulesuivante.Value = Cellulecourante.Value Then
14:          If LignesIdentiques(Cellulecourante, Cellulesuivante, 3) = True Then
15:             Cellulecourante.EntireRow.Delete
16:          End If
17:       End If
18:       Set Cellulecourante = Cellulesuivante
19:    Loop
20: End Sub
21:
22: Function LignesIdentiques(CellCourante As Range, Cellsuivante As Range,
      ➥ Num As Byte) As Boolean
23:    LignesIdentiques = True
24:    Dim compteur As Byte
25:    'boucle et test des Num colonnes
26:    For compteur = 1 To Num
27:       If CellCourante.Offset(0, compteur).Value <> Cellsuivante.Offset(0,
            ➥ compteur).Value Then
28:          LignesIdentiques = False
29:          Exit For
30:       End If
31:    Next compteur
32: End Function
```

Ligne 14, la fonction `LignesIdentiques` est appelée et reçoit maintenant une valeur de type `Byte` pour l'argument *`Num`* (ici, 3).

La fonction `LignesIdentiques` contrôle ensuite *`Num`* cellules afin de définir si la ligne doit ou non être supprimée. Elle reçoit d'abord la valeur `True`. La boucle `For…Next` (lignes 26 à 31) s'exécute ensuite *`Num`* fois. Les cellules testées à chaque passage de la boucle correspondent à un déplacement de *`Num`* cases vers la droite. Si deux contenus différents sont décelés (ligne 27), la valeur `False` est affectée à `LignesIdentiques` et l'instruction `Exit For` entraîne la sortie de la boucle. Si les contenus des cellules comparées sont toujours identiques, la boucle prend fin après `Num` passages et la fonction garde la valeur `True`.

Boucle For…Next avec pas négatif

Le programme suivant supprime les lignes vides de la feuille active. Il utilise pour ce faire une structure `For…Next` avec un pas négatif de –1, de façon à parcourir l'ensemble des lignes de la feuille, de la dernière ligne employée jusqu'à la première.

```
1: Sub SupprLignesVides()
2:     Dim DerniereLigne As Long
```

```
3:     Dim Compteur As Long
4:     DerniereLigne = ActiveSheet.UsedRange.Row - 1 _
                  + ActiveSheet.UsedRange.Rows.Count
5:     For Compteur = DerniereLigne To 1 Step -1
6:        If Application.WorksheetFunction.CountA(Rows(Compteur))=0 _
            Then Rows(Compteur).Delete
7:     Next Compteur
8: End Sub
```

Lignes 2 et 3, les variables sont déclarées. Ligne 4, on affecte à `DerniereLigne` le numéro de la dernière ligne employée. On se sert pour cela de la propriété `UsedRange` qui renvoie la zone utilisée sur la feuille active, c'est-à-dire la zone rassemblant l'ensemble des cellules contenant des données sur la feuille. La propriété `Row` renvoie le numéro de la première ligne de cette zone. En retirant 1 à cette valeur, on obtient le nombre de lignes vides précédant la zone utilisée. L'expression `UsedRange.Rows.Count` renvoie le nombre de lignes de la zone. En additionnant ces deux valeurs, nous obtenons le numéro de la dernière ligne de la zone utilisée.

Une boucle `For…Next` avec un pas négatif (`Step -1`) est ensuite employée pour parcourir les lignes à vérifier en commençant par la dernière. Ligne 6, on vérifie si la ligne testée est vide à l'aide de la fonction Excel `CountA` qui renvoie le nombre de cellules de la zone interrogée (ici la ligne entière, `Rows(Compteur)`) contenant des données. Si la ligne est vide (`CountA` renvoie 0), elle est supprimée. Le programme passe à la valeur suivante, en décrémentant notre compteur de 1.

L'utilisation d'un pas négatif nous assure ici que le programme teste toutes les lignes. En effet, si nous avions employé un pas positif et parcouru la zone utilisée de la première ligne à la dernière, la suppression d'une ligne aurait entraîné le décalage vers le haut de la suivante, qui n'aurait donc pas été traitée lors du prochain passage de la boucle.

Boucles For…Next imbriquées

Il est possible d'imbriquer des instructions `For…Next`. Veillez simplement à donner des noms différents à chacune des variables compteurs.

La procédure suivante utilise une structure `For…Next` pour stocker les valeurs du tableau présenté à la figure 7-7 dans une variable de matrice multidimensionnelle.

	A	B	C	D	E	F
1		Livres	Vidéo	Hi-Fi	Autres	
2	Janvier	58 963,00	45 225,00	85 485,00	45 225,00	
3	Février	45 895,00	32 568,00	79 658,00	32 568,00	
4	Mars	69 785,00	46 895,00	25 689,00	46 895,00	
5	Avril	45 214,00	54 897,00	49 652,00	54 897,00	
6	Mai	45 258,00	32 568,00	36 550,00	32 568,00	
7	Juin	38 652,00	97 632,00	56 320,00	97 632,00	
8	Juillet	32 510,00	45 863,00	45 520,00	45 863,00	
9	Août	28 952,00	32 568,00	47 965,00	32 568,00	
10	Septembre	45 693,00	94 625,00	29 865,00	94 625,00	
11	Octobre	48 956,00	31 582,00	16 495,00	31 582,00	
12	Novembre	65 920,00	21 458,00	75 632,00	21 458,00	
13	Décembre	95 120,00	12 589,00	12 589,00	12 589,00	
14						

Figure 7-7 – *Pour stocker les valeurs d'une feuille Excel dans une variable de matrice, utilisez des boucles For…Next imbriquées.*

```
Dim MonTableau() As Single


 1: Sub BouclesForNextImbriquées()
 2:   Dim DerniereLigne As Byte
 3:   DerniereLigne = Range("A2").End(xlDown).Row
 4:   Dim NbreDeLignes As Byte
 5:   NbreDeLignes = DerniereLigne - 1
 6:   ReDim MonTableau(NbreDeLignes,4)
 7:   Call AffecterValeursTableau(NbreDeLignes)
 8: End Sub


 9: Sub AffecterValeursTableau(DerniereLigneTableau)
10:   Dim CompteurLignes As Byte
11:   Dim CompteurColonnes As Byte
12:   For CompteurLignes = 1 To DerniereLigneTableau
13:      For CompteurColonnes = 1 To 4
14:         MonTableau(CompteurLignes, CompteurColonnes) = _
              Cells(CompteurLignes+1, CompteurColonnes+1)
15:      Next CompteurColonnes
16:   Next CompteurLignes
17: End Sub
```

La variable de matrice `MonTableau()` est déclarée dans la section Déclarations du module, afin d'être accessible à toutes les procédures de ce dernier. La procédure `BouclesForNextImbriquées` la redimensionne de sorte qu'elle accueille l'ensemble des données de la feuille Excel active. Pour un descriptif des instructions de cette procédure, reportez-vous à la section « Variables de matrice dynamiques » du chapitre précédent. Elle appelle ensuite la procédure `AffecterValeursTableau` en lui passant la valeur de la variable `NbreDeLignes`.

La procédure `AffecterValeursTableau` commence par créer deux variables

numériques de type `Byte`, qui serviront de compteur à chacune des boucles `For…Next` (lignes 10 et 11). La première boucle `For…Next` (lignes 12 à 16) utilise `CompteurLignes` pour répéter son instruction autant de fois qu'il y a de lignes à stocker. La boucle `For…Next` imbriquée (lignes 13 à 15) utilise `CompteurColonnes` afin de répéter ses instructions pour chacune des colonnes. L'instruction ligne 14 affecte une valeur à un des espaces de stockage de `MonTableau`, selon l'ordre suivant :

1. CompteurLigne = 1 : la boucle `For…Next` imbriquée s'exécute quatre fois et affecte les valeurs des cellules B2 à E2 à `MonTableau(1,1)`, `MonTableau(1,2)`, `MonTableau(1,3)` et `MonTableau(1,4)`.
2. CompteurLigne = 2 : la boucle `For…Next` imbriquée s'exécute quatre fois et affecte les valeurs des cellules B3 à E3 à `MonTableau(2,1)`, `MonTableau(2,2)`, `MonTableau(2,3)` et `MonTableau(2,4)`.

   [etc.]

La boucle For Each…Next

Cette structure de contrôle généralise un traitement à l'ensemble des objets d'une collection et s'utilise selon la syntaxe suivante :

```
For Each élément In Collection
   Instructions
Next élément
```

où *élément* est une variable de type `Object` ou `Variant`, utilisée pour représenter chaque objet de la collection. Les *Instructions* sont exécutées une fois pour chaque objet de la collection. La procédure suivante utilise une structure `For Each…Next` pour appliquer une couleur de police rouge (`ColorIndex = 3`) à tous les objets `Cells` de l'objet `Selection` (toutes les cellules de la sélection en cours) dont la valeur est supérieure à 1 000.

```
Sub FortesValeursEnRouge()
  Dim cellule As Range
    For Each cellule In Selection.Cells
      If cellule.Value>1000 Then
         cellule.Font.ColorIndex = 3
      End If
    Next cellule
End Sub
```

La procédure suivante enregistre tous les classeurs Excel ouverts (au format xlsx) – à l'exception de PERSONAL.XLSB – vers le format Excel 97-2003 (xls) et les ferme :

```
 1: Sub EnregistrerFormatExcel4EtFermer()
```

```
 2:   Dim Classeur As Workbook
 3:   Dim position As Byte
 4:   Dim NomClasseur As String
 5:     For Each Classeur In Workbooks
 6:       If Not Classeur.Name="PERSONAL.XLSB" Then
 7:         NomClasseur = Classeur.FullName
 8:         position = InStr(NomClasseur, ".xlsx")
 9:         NomClasseur = Left(NomClasseur, position-1) & ".xls"
10:         Classeur.SaveAs FileName:=NomClasseur, _
             FileFormat:=xlExcel8
11:         Classeur.Close
12:       End If
13:     Next Classeur
14: End Sub
```

Les lignes 2 à 4 déclarent les variables nécessaires au programme. La condition de la ligne 6 vérifie que `Classeur` n'est pas PERSONAL.XLSB. On utilise pour de faire l'opérateur logique `Not` (présenté à la fin de ce chapitre) et la propriété `Name` qui, attachée à un objet `Workbook`, renvoie le nom de fichier de ce dernier (sans le chemin).

Ligne 7, la variable `NomClasseur` se voit affecter le nom complet du classeur – le chemin suivi du nom de fichier – renvoyé par la propriété `FullName`. Les fonctions `InStr` et `Left` sont utilisées pour substituer l'extension .xls à l'extension .xlsx, afin de définir des noms d'enregistrement corrects pour les fichiers. `InStr` sert à comparer deux chaînes de caractères. Ici, elle renvoie une valeur numérique représentant la position de ".xlsx" dans la chaîne `NomClasseur`. La fonction `Left` sert à renvoyer un nombre déterminé de caractères situés à gauche d'une chaîne. Ici, elle renvoie les (position–1) premiers caractères de la chaîne `NomClasseur`, c'est-à-dire le nom du fichier sans l'extension ".xlsx". Il suffit alors de concaténer la valeur de `NomClasseur` et la chaîne ".xls" pour obtenir un nom du fichier à sauvegarder. La méthode `SaveAs` est ensuite appliquée à l'objet `Classeur`, le nom ainsi défini est affecté à l'argument `FileName`, et la constante Excel `xlExcel8` à l'argument `FileFormat`. Enfin, la méthode `Close` ferme le document ainsi enregistré.

Info

Le traitement des chaînes de caractères est un sujet incontournable. Vous serez inévitablement amené à manipuler des chaînes (composées de lettres comme de chiffres) afin d'en extraire les données voulues ou de les modifier. Les fonctions de traitement des chaînes de caractères sont présentées au chapitres 11.

Boucles For Each…Next imbriquées

À l'instar des boucles `For…Next` et, plus largement, de l'ensemble des structures de contrôle, vous pouvez imbriquer des structures de contrôle `For Each…Next`. L'exemple suivant extrait l'ensemble des formules du classeur inscrit et les

écrit dans un document Word qui est ensuite imprimé. Nous utilisons pour ce faire deux structures `For Each…Next`. La première parcourt la collection des feuilles de travail (`ActiveWorkbook.Worksheets`) du classeur, tandis que la seconde y est imbriquée et parcourt la collection des cellules de la zone courante définie à partir de la cellule A1 (`MaFeuille.Cells(1,1).CurrentRegion.Cells`). Lorsque la boucle imbriquée a fini de traiter les cellules de la zone courante de la feuille de travail en cours, la première structure `For Each…Next` reprend la main et traite donc l'objet `Worksheet` – la feuille de travail – suivant de la collection.

```
 1: Public Sub ExtraireMesFormulesWord()
 2:   Dim MaFormule As String
 3:   Dim MaCellule As Range
 4:   Dim MaFeuille As Worksheet
 5:   Dim MonWord As Object
 6:   On Error Resume Next
 7:     Set MonWord = GetObject(, "Word.Application")
 8:   If Err.Number<>0 Then
 9:     Set MonWord = CreateObject("Word.Application")
10:     Err.Clear
11:   End If
12:   MonWord.Visible = True
13:   MonWord.Documents.Add

14:   For Each MaFeuille In ActiveWorkbook.Worksheets
15:     With MonWord.Selection
16:       .Font.Name = "Arial"
17:       .Font.Bold = True
18:       .Font.Size = "13"
19:       .TypeText "Formules de la feuille : " & MaFeuille.Name & Chr(13)
20:       .Font.Size = "11"
21:       .Font.Bold = False
22:     End With
23:     For Each MaCellule In MaFeuille.Cells(1,1).CurrentRegion.Cells
24:       If MaCellule.HasFormula=True Then
25:         MaFormule = "{" & MaCellule.Formula & "}"
26:         MonWord.Selection.TypeText "Cellule " & _
27:           MaCellule.Address(False, False, xlA1) & _
28:           " : " & MaFormule & Chr(13)
29:       End If
30:     Next MaCellule
31:   Next MaFeuille
32: End Sub
```

Lignes 2 à 5, les variables sont déclarées. Lignes 6 à 13, `MonWord` reçoit l'objet `Word.Application`. Notez que la méthode `GetObject` est utilisée avec un gestionnaire d'erreurs afin de capturer l'erreur générée si Word n'est pas ouvert, auquel cas la méthode `CreateObject` crée une nouvelle instance de l'application (ligne 9) et l'objet `Err` qui reçoit l'erreur est réinitialisé (ligne 10). Lignes 12 et 13, l'application Word s'affiche et un nouveau document est créé.

Lignes 14 à 31, la première boucle `For Each…Next` parcourt la collection des feuilles du classeur. Pour chacune, le document Word reçoit un texte formaté (lignes 15 à 22). Nous utilisons pour cela une structure `With…End With` qui définit

la police employée (Arial, corps 13), avant d'insérer le texte "Formules de la feuille : " suivi du nom de la feuille. La taille de la police est ensuite redéfinie à 11.

La seconde structure `For Each…Next` (lignes 23 à 30) prend alors la main et traite chacune des cellules de la collection `MaFeuille.Cells(1,1).CurrentRegion.Cells`. Notez que l'on utilise ici la propriété `CurrentRegion` pour définir la zone courante à partir de la cellule A1 de la feuille. À chaque occurrence de la boucle, on vérifie si la cellule contient une formule (ligne 24) et, si tel est le cas, on l'insère dans le document Word, précédée de l'adresse de la cellule concernée.

Lorsque toutes les cellules ont été traitées, la boucle imbriquée prend fin et l'instruction de la ligne 31 appelle le passage suivant de la structure `For Each… Next` principale.

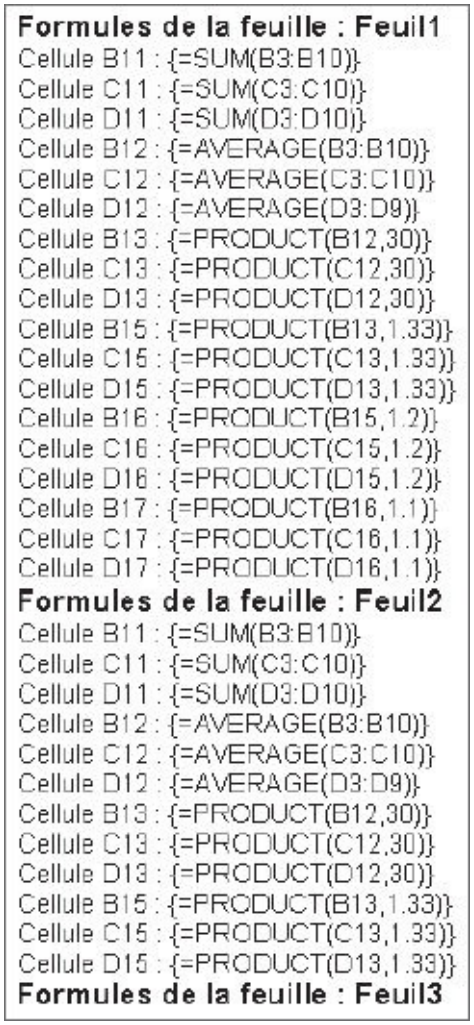

Formules de la feuille : Feuil1
Cellule B11 : {=SUM(B3:B10)}
Cellule C11 : {=SUM(C3:C10)}
Cellule D11 : {=SUM(D3:D10)}
Cellule B12 : {=AVERAGE(B3:B10)}
Cellule C12 : {=AVERAGE(C3:C10)}
Cellule D12 : {=AVERAGE(D3:D9)}
Cellule B13 : {=PRODUCT(B12,30)}
Cellule C13 : {=PRODUCT(C12,30)}
Cellule D13 : {=PRODUCT(D12,30)}
Cellule B15 : {=PRODUCT(B13,1.33)}
Cellule C15 : {=PRODUCT(C13,1.33)}
Cellule D15 : {=PRODUCT(D13,1.33)}
Cellule B16 : {=PRODUCT(B15,1.2)}
Cellule C16 : {=PRODUCT(C15,1.2)}
Cellule D16 : {=PRODUCT(D15,1.2)}
Cellule B17 : {=PRODUCT(B16,1.1)}
Cellule C17 : {=PRODUCT(C16,1.1)}
Cellule D17 : {=PRODUCT(D16,1.1)}
Formules de la feuille : Feuil2
Cellule B11 : {=SUM(B3:B10)}
Cellule C11 : {=SUM(C3:C10)}
Cellule D11 : {=SUM(D3:D10)}
Cellule B12 : {=AVERAGE(B3:B10)}
Cellule C12 : {=AVERAGE(C3:C10)}
Cellule D12 : {=AVERAGE(D3:D9)}
Cellule B13 : {=PRODUCT(B12,30)}
Cellule C13 : {=PRODUCT(C12,30)}
Cellule D13 : {=PRODUCT(D12,30)}
Cellule B15 : {=PRODUCT(B13,1.33)}
Cellule C15 : {=PRODUCT(C13,1.33)}
Cellule D15 : {=PRODUCT(D13,1.33)}
Formules de la feuille : Feuil3

Figure 7-8 – *Toutes les formules du classeur ont été extraites.*

Utiliser des instructions conditionnelles

Si les boucles permettent de réaliser des macros puissantes, l'usage des conditions leur assurera souplesse et sûreté en :

- garantissant que l'environnement de l'application et l'état du document sont compatibles avec l'exécution du programme ;
- modifiant le comportement de la macro selon l'état du document et de l'application à un moment précis ;
- échangeant des informations avec l'utilisateur lors de l'exécution du programme (combinées avec la fonction `MsgBox`, par exemple).

La structure de contrôle If…Then…Else

`If…Then...Else` est également une instruction conditionnelle. Cependant, elle est plus souple et plus répandue que `While…Wend`, qui est essentiellement utilisée pour réaliser des boucles. La structure `If…Then…Else` spécifie en effet différentes options d'exécution dans une procédure, en fonction de l'état de l'application ou/et du document. En outre, elle conjugue les conditions dans des boucles imbriquées.

Dans sa forme minimale, l'instruction conditionnelle `If` se présente ainsi :

```
If Condition Then
   Série d'instructions
End If
```

Lorsque la *`Condition`* spécifiée est remplie, la *`Série d'instructions`* est exécutée ; sinon, la procédure se poursuit avec l'instruction située immédiatement après `End If`.

La macro `Auto_Open` suivante utilise une structure `If…End If` pour contrôler l'affichage de la boîte de dialogue présentée à la figure 7-9. Elle recourt pour ce faire à la fonction `Date`, qui renvoie la date du jour.

> **Rappel**
>
> Une macro `Auto_Open` s'exécute automatiquement à l'ouverture du classeur Excel dans lequel elle est stockée.

```
Sub Auto_Open()
  If Date>"30/11/17" and Date<"08/12/17" Then
    MsgBox "Attention ! Remise des budgets " & _
      "prévisionnels le 8 décembre.", _
      vbOKOnly + vbCritical, "Soyez prêt !"
  End If
```

```
End Sub
```

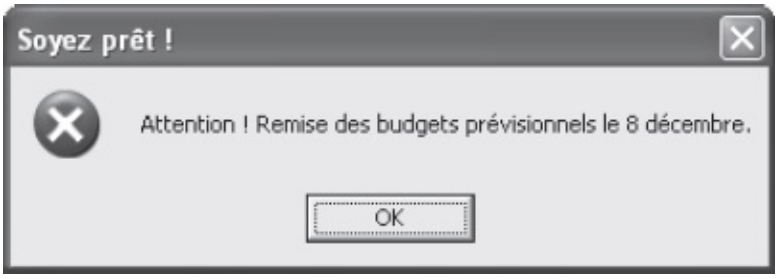


Figure 7-9 – *Ce message s'affiche à chaque ouverture du classeur effectuée entre le 1er et le 7 décembre.*

La valeur attachée à l'opérateur relationnel dans une condition varie selon l'objet de la comparaison ; il peut s'agir d'une chaîne de caractères, d'un nombre, ou encore d'une valeur booléenne.

Dans l'exemple suivant, l'instruction `If` assure que les conditions nécessaires au bon fonctionnement du programme sont réalisées (en l'occurrence que deux fenêtres de document sont ouvertes). Si ce n'est pas le cas, un message s'affiche à l'attention de l'utilisateur et l'instruction `Exit Sub` entraîne la sortie de la procédure.

```
Sub VérifierConditions
  If Workbooks.Count<>2 Then
    MsgBox "La macro ne peut être exécutée. " & _
    "Deux classeurs doivent être ouverts."
    Exit Sub
  End if
  'Instructions exécutées si deux classeurs sont ouverts
End Sub
```

Une structure `If…Then…Else` autorise un nombre indéterminé de conditions. Vous pouvez ainsi envisager les différents cas possibles dans une situation particulière et indiquer à la procédure les instructions à exécuter dans chacun de ces cas.

L'instruction répond alors à la syntaxe suivante :

```
If condition1 Then
  Série d'instructions 1
ElseIf condition2 Then
  Série d'instructions 2
ElseIf condition3 Then
  Série d'instructions 3
…
Else
  Série d'instructions n
End If
```

Contrairement à `ElseIf`, l'instruction `Else` ne pose aucune condition : elle

apparaît en dernière position, et les instructions qui lui sont attachées sont automatiquement exécutées si aucune des conditions posées auparavant n'a été réalisée. En revanche, si l'une des conditions posées par une instruction `If` ou `ElseIf` est réalisée, la macro exécute les instructions qui lui sont attachées, puis ignore tout le reste et se poursuit avec les instructions situées après `End If`.

L'instruction `ElseIf`, comme `Else`, est facultative. Une instruction conditionnelle peut être composée d'une ou de plusieurs instructions `ElseIf` et ne pas présenter d'instruction `Else`, et inversement. La fonction suivante détermine la valeur d'une remise sur un achat, puis insère cette valeur ainsi que le prix après remise dans la feuille de calcul.

```
 1: Sub CalculRemiseEtPrixDefinitif()
 2:   Dim PrixAvantRemise As Single, PrixDefinitif As Single
 3:   PrixAvantRemise = ActiveSheet.Range("C11")
 4:   PrixDefinitif = PrixAvecRemise(PrixAvantRemise)
 5:   ActiveSheet.Range("C13").Value = PrixDefinitif
 6: End Sub


 7: Function PrixAvecRemise(ValeurAchat)
 8:   Dim PourcentageRemise As Single
 9:   If ValeurAchat<=1000 Then
10:     PourcentageRemise = 0
11:   ElseIf ValeurAchat>1000 And ValeurAchat<=2000 Then
12:     PourcentageRemise = 0.1
13:   ElseIf ValeurAchat>2000 And ValeurAchat<=5000 Then
14:     PourcentageRemise = 0.2
15:   ElseIf ValeurAchat>5000 And ValeurAchat<10000 Then
16:     PourcentageRemise = 0.25
17:   Else
18:     PourcentageRemise = 0.3
19:   End If
20:   ActiveSheet.Range("C12").Value = PourcentageRemise
21:   PrixAvecRemise = ValeurAchat - (ValeurAchat * PourcentageRemise)
22: End Function
```

La procédure `CalculRemiseEtPrixDefinitif` déclare les variables `PrixAvantRemise` et `PrixDefinitif` de type `Single`. `PrixAvantRemise` reçoit la valeur de la cellule C11 de la feuille active. L'instruction de la ligne 4 appelle la fonction `PrixAvecRemise` en lui passant cette valeur.

La structure conditionnelle `If…Then…Else` des lignes 9 à 19 détermine `PourcentageRemise` en fonction de `ValeurAchat`, et la valeur de la remise est insérée dans la cellule C12. Ligne 21, la fonction reçoit la valeur après remise, c'est-à-dire la valeur de `ValeurAchat` moins le prix de la remise (`ValeurAchat * PourcentageRemise`).

La procédure principale reprend ensuite la main. L'instruction de la ligne 5 affecte alors à la cellule C13 la valeur de `PrixDefinitif`. La procédure prend fin.

Info

Une instruction conditionnelle peut aussi s'écrire sur une seule ligne, en utilisant deux points (:) comme séparateurs entre les instructions à exécuter si la condition est vérifiée. L'instruction `End If` est alors omise :

```
If Condition Then Instruction1 : Instruction2 : … : InstructionN
```

Par exemple, l'instruction :

```
If Selection.Font.Italic()=True Then
  Selection.Font.Italic()=False
End If
```

est aussi valide sous la forme :

```
If Selection.Font.Italic()=True Then Selection.Font.Italic()=False
```

Voici un exemple avec plusieurs instructions sur une même ligne :

```
If Selection.Font.Italic()=True Then Selection.Font.Italic()=False :
    ➥Selection.Font.Bold=True
```

Conditions imbriquées

Les conditions imbriquées permettent de prendre en considération un grand nombre de possibilités lors de l'exécution du programme.

L'exemple suivant est composé d'une première instruction conditionnelle qui vérifie si deux fenêtres sont ouvertes avant de s'exécuter. Nous y avons *imbriqué* une instruction conditionnelle, qui modifie la boîte de dialogue affichée en fonction du nombre de fenêtres ouvertes.

L'organigramme de la figure 7-10 présente la structure de cette macro.

```
Sub ConditionsImbriquées()
  If Workbooks.Count<>2 Then
    Dim Message As String
    If Workbooks.Count<2 Then
      Message = "Au moins deux documents doivent être ouverts."
    Else
      Message = "Seuls les deux documents concernés doivent être ouverts."
    End If
    MsgBox Message, vbOKOnly + vbInformation, "Exécution impossible"
    Exit Sub
  End if
  Instructions de la macro
End Sub
```

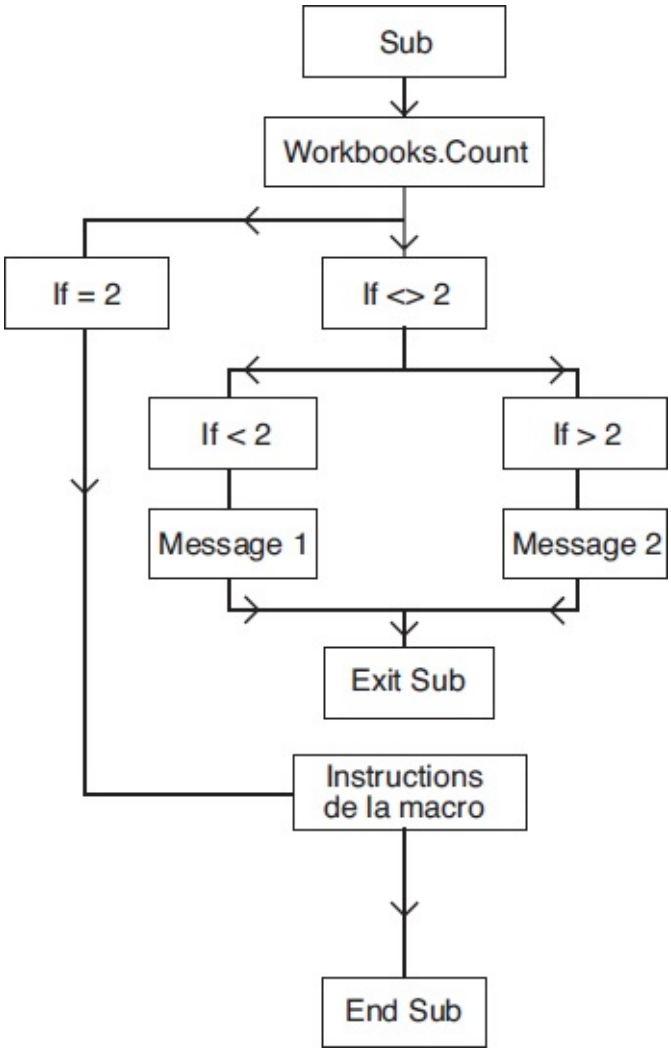


Figure 7-10 – *Les conditions imbriquées assurent aux macros souplesse et précision.*

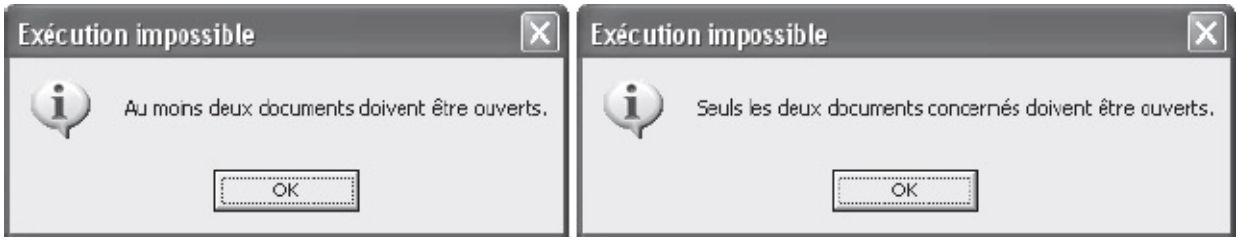


Figure 7-11 – *L'instruction conditionnelle imbriquée détermine le message qui sera affiché.*

La structure de contrôle Select Case

La structure de contrôle `Select Case` permet d'envisager différentes valeurs pour

une même expression et de déterminer des instructions spécifiques pour chaque cas. Elle répond à la syntaxe suivante :

```
Select Case Expression
  Case valeur1
    Instructions
  Case valeur2
    Instructions
   …
  Case valeurn
    Instructions
  Case Else
    Instructions
End Select
```

Lorsque la valeur renvoyée par *Expression* correspond à l'une de celles posées par les instructions `Case`, les *Instructions* correspondantes s'exécutent et la procédure se poursuit avec l'instruction qui suit `End Select`. Si aucune des valeurs ne correspond, les instructions attachées à `Case Else`, s'il existe, sont exécutées.

Le programme suivant détermine la valeur de la variable `Reduction`, selon le contenu de la cellule D7 de la feuille 1 dans le classeur commande.xlsx (notez que celui-ci doit être ouvert au moment de l'exécution de la macro). Une boîte de dialogue est ensuite affichée, afin d'informer l'utilisateur de la remise qui sera effectuée.

```
 1: Sub AffichageReduction()
 2:   Dim Reduction As Variant
 3:   Dim LongueurChaîne As Byte
 4:   Reduction = CalculerValeurReduction _
       (Workbooks("commande.xlsx").Sheets(1).Range("D7").Value)
 5:   LongueurChaîne = Len(Reduction)
 6:   If LongueurChaîne=3 Then Reduction = Reduction & "0"
 7:   MsgBox "La remise effectuée sera de " & Reduction & " %."
 8: End Sub


 9: Function CalculerValeurReduction(PrixCommande)
10:   Select Case PrixCommande
11:     Case 0 To 999.99
12:       CalculerValeurReduction = 0
13:     Case 1000 To 1999.99
14:       CalculerValeurReduction = 0.1
15:     Case 2000 To 2999.99
16:       CalculerValeurReduction = 0.25
17:     Case Else
18:       CalculerValeurReduction = 0.4
19:   End Select
20: End Function
```

La variable `Reduction` est tout d'abord déclarée de type `Variant`. Elle stockera en effet une valeur numérique, qui sera ensuite manipulée en tant que chaîne de caractères. L'instruction d'affectation de la ligne 4 appelle la fonction `CalculerValeurReduction`, en lui passant la valeur de la cellule D7 de la première

feuille du classeur commande.xlsx.

La fonction `CalculerValeurReduction` utilise une structure `Select Case` pour renvoyer une valeur fonction de l'argument `PrixCommande` (ici la valeur de la cellule D7). Le mot-clé `To` est utilisé pour définir des plages de valeurs (0 à 999.99, 1000 à 1999.99, etc.). La procédure principale reprend ensuite la main.

Les instructions des lignes 5 et 6 servent à formater la chaîne stockée dans la variable `Reduction`. La fonction `Len` renvoie la longueur (le nombre de caractères) de `Reduction`, qui est stockée dans `LongueurChaîne`. Si la chaîne comprend trois caractères, un 0 est ajouté à la fin (0,1 et 0,4 deviennent respectivement 0,10 et 0,40). Enfin, l'instruction de la ligne 7 affiche une boîte de dialogue informant l'utilisateur de la valeur de la remise qui sera effectuée.

Info

Dans les instructions Visual Basic, c'est le point qui sert de séparateur décimal dans les valeurs numériques. Cependant, lorsque vous affichez une valeur numérique sous forme de chaîne – comme l'instruction de la ligne 7 de l'exemple précédent –, la virgule est utilisée.

Définir l'instruction suivante avec GoTo

L'instruction `GoTo` oriente le déroulement d'une procédure vers l'emplacement spécifié par l'utilisateur, à tout moment de l'exécution. Cette instruction s'utilise avec une étiquette, c'est-à-dire une balise placée dans le texte. La syntaxe de `GoTo` est la suivante :

```
GoTo Etiquette
```

L'étiquette spécifiée après `GoTo` doit être placée au début d'une ligne indépendante, située avant l'instruction sur laquelle on veut brancher la procédure ; elle doit être immédiatement suivie des deux points « : ». Une instruction `GoTo` ne peut renvoyer qu'à une étiquette se trouvant dans la même procédure.

Les instructions `GoTo` compliquent la lecture du code. Préférez-leur les structures de contrôle.

Interagir avec l'utilisateur via des boîtes de dialogue

L'affichage de boîtes de dialogue au cours de l'exécution d'un programme renseigne l'utilisateur sur son déroulement, ou lui demande des informations qui en modifieront le cours. Deux fonctions affichent des boîtes de dialogue :

- **`InputBox`** entraîne l'affichage d'une boîte de dialogue présentant une zone de texte dans laquelle l'utilisateur est invité à entrer des informations.
- **`MsgBox`** affiche un message à l'attention de l'utilisateur et lui propose de choisir entre différentes possibilités en cliquant sur l'un des boutons de commande affichés.

La fonction InputBox

La fonction VBA `InputBox` affiche une boîte de dialogue contenant une zone de texte légendée, afin d'inviter l'utilisateur à y saisir l'information attendue ; cette dernière est renvoyée sous forme de chaîne de caractères et stockée dans une variable pour être ensuite exploitée par le programme.

La fonction `InputBox` s'utilise selon la syntaxe suivante :

```
InputBox(prompt, title, default)
```

`prompt`, *`title`* et *`default`* sont des arguments nommés de type `String`. *`prompt`* est le message affiché dans la boîte de dialogue afin de légender la zone de texte. L'argument *`title`*, facultatif, correspond au texte affiché dans la barre de titre de la boîte de dialogue ; s'il est omis, c'est le nom de l'application qui apparaît. *`default`* est facultatif aussi ; il définit ce qui apparaît par défaut dans la zone de texte.

La procédure suivante affiche la boîte de dialogue représentée à la figure 7-12.

```
Sub UtilisationDeInputBox()
  Dim DateVal As String
  DateVal = InputBox("Date de validité :", "Nouveau membre", Date)
End Sub
```

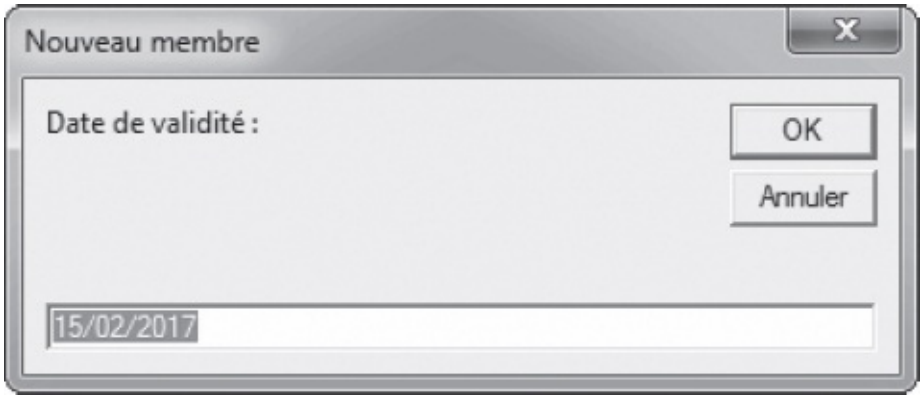


Figure 7-12 – *La fonction InputBox permet de demander à l'utilisateur d'entrer des données.*

Pour stocker l'information fournie par l'utilisateur, il suffit d'affecter la fonction à une variable de type `String` ou `Variant`, selon la syntaxe suivante :

```
Variable = InputBox(prompt, title, default)
```

Conseil

Si l'affichage d'une réponse par défaut n'est pas indispensable, il peut se révéler fort pratique pour orienter l'utilisateur lorsque l'information demandée autorise plusieurs formats. Par exemple, si vous avez besoin d'une date au format jj/mm/aa, l'affichage d'une date hypothétique à l'aide de l'argument `default` indiquera à l'utilisateur le format attendu.

Vous pouvez afficher des variables ou des caractères réservés (comme le guillemet ou la virgule) dans une boîte de dialogue. Pour ce faire, on utilise :

- l'opérateur de concaténation `&` ;
- la fonction `Chr`, pour afficher un caractère réservé en spécifiant son code ASCII (entre 1 et 32).

La fonction `InputBox` se présente alors ainsi :

```
InputBox("Texte" & variable + Chr(CodeAscii) + "Texte", "Titre", "Entrée par défaut")
```

La procédure suivante affiche la boîte de dialogue présentée à la figure 7-13 :

```
Sub OuvertureSession()
  Dim Utilisateur As String, MotDePasse As String
  Utilisateur = Application.UserName
  MotDePasse = InputBox("Utilisateur : " & Utilisateur & Chr(10) & _
    "Veuillez entrer votre mot de passe.", "Saisie du mot de passe")
  Call VerifierMotDePasse(Utilisateur, MotDePasse)
End Sub
```

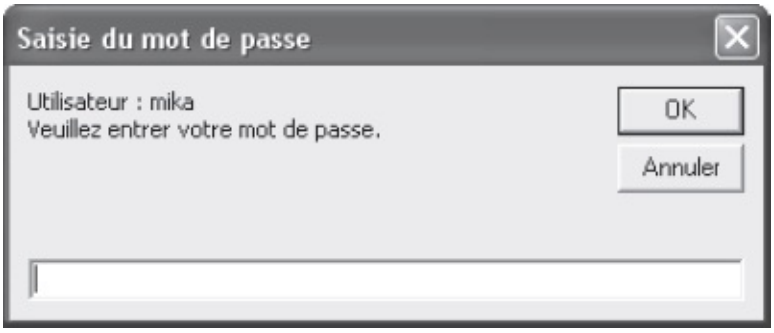


Figure 7-13 – *Utilisez l'opérateur de concaténation & pour intégrer des variables ou des caractères réservés dans les chaînes de caractères.*

Dans l'exemple suivant, la fonction `InputBox` demande à l'utilisateur d'indiquer une date d'échéance pour les opérations en cours. La procédure met ensuite en valeur les cellules sélectionnées dont la date est supérieure à la date indiquée.

```
 1: Sub VerifierEcheances()
 2:   'Vérifier qu'il existe une plage de cellules sélectionnée
 3:   Dim ZoneATester As String
 4:   ZoneATester = ActiveWindow.RangeSelection.Address
 5:   If ZoneATester=Null Then
 6:     MsgBox "Sélectionnez la plage de cellules à tester.", _
         vbOKOnly + vbInformation
 7:   Exit Sub
 8:   End If


 9:   'Demander à l'utilisateur la date d'échéance
10:   Dim DateEcheance As Variant
11:   DateEcheance = InputBox("Indiquez la date d'échéance.", _
       "Echéance des opérations en cours ", Date + 30)
12:   DateEcheance = CDate(DateEcheance)
13:   'Tester toutes les cellules de la sélection
14:   Dim CellTest As Range
15:   For Each CellTest In Range(ZoneATester)
16:     If IsDate(CellTest)=True Then
17:       If CellTest.Value>DateEcheance Then
18:         CellTest.Interior.ColorIndex = 6
19:       End If
20:     End If
21:   Next
22: End Sub
```

La procédure commence par vérifier qu'une plage de cellules a été sélectionnée dans le classeur actif (lignes 2 à 8). Elle affecte pour cela l'adresse de la sélection en cours à la variable `ZoneATester`. Si aucune zone n'est sélectionnée (`ZoneATester=Null`), un message s'affiche à l'attention de l'utilisateur et l'instruction de la ligne 7 entraîne la sortie de la procédure.

La fonction `InputBox` demande ensuite à l'utilisateur d'indiquer une date d'échéance (ligne 11). La fonction `Date` est utilisée pour déterminer la valeur par défaut de la zone de texte. Elle renvoie la date du jour, à laquelle on ajoute 30 jours. La chaîne renvoyée par la fonction `InputBox` est stockée dans la variable `DateEcheance` de type `Variant`. Ligne 12, la fonction `CDate` convertit `DateEcheance` en une variable de type `Date`.

Attention

Si vous déclarez `DateEcheance` de type `String`, le programme fonctionnera correctement, mais l'instruction de conversion de type de données de la ligne 12 ne modifiera rien. L'instruction conditionnelle de la ligne 17 effectuera alors une comparaison entre les chaînes de caractères, et non entre les dates. Le programme se déroulera correctement, mais produira des résultats erronés.

La procédure teste ensuite l'ensemble des cellules sélectionnées. Une variable objet de type `Range` est déclarée ligne 14. Lignes 15 à 22, une structure de contrôle `For Each…Next` est utilisée pour tester tous les objets `Range` de la collection contenue dans la sélection en cours.

L'instruction conditionnelle de la ligne 16 vérifie que la cellule traitée contient

des données de type `Date`. Si ce n'est pas le cas, elle est ignorée et la boucle se poursuit avec la cellule suivante. Si les données sont de type `Date`, la structure conditionnelle des lignes 17 à 19 est exécutée : quand la valeur de la cellule est supérieure à `DateEcheance`, elle est peinte en jaune (`ColorIndex=6`). La figure 7-14 représente une feuille de calcul après passage de la macro `VerifierEcheances` (l'utilisateur a indiqué le 16/03/2017 pour date d'échéance).

	A	B	C	D	E	F
1	20/02/2017	20/02/2017	20/02/2017	05/02/2017	08/03/2017	08/03/2017
2	22/02/2017	22/02/2017	22/02/2017	10/02/2017	09/03/2017	09/03/2017
3	24/02/2017	14/04/2017	15/04/2017	15/02/2017	10/03/2017	10/03/2017
4	26/02/2017	26/02/2017	06/06/2017	20/02/2017	11/03/2017	11/03/2017
5	28/02/2017	28/02/2017	28/07/2017	25/02/2017	12/03/2017	12/03/2017
6	02/03/2017	02/03/2017	18/09/2017	02/03/2017	13/03/2017	13/03/2017
7	04/03/2017	05/03/2017	06/03/2017	07/03/2017	14/03/2017	09/03/2017
8	06/03/2017	07/03/2017	08/03/2017	08/03/2017	15/03/2017	10/03/2017
9	08/03/2017	02/03/2017	10/03/2017	10/03/2017	16/03/2017	11/03/2017
10	10/03/2017	05/03/2017	12/03/2017	12/03/2017	17/03/2017	12/03/2017
11	12/03/2017	07/03/2017	14/03/2017	14/03/2017	18/03/2017	13/03/2017
12	14/03/2017	02/03/2017	16/03/2017	16/03/2017	19/03/2017	20/03/2017
13	16/03/2017	05/03/2017	18/03/2017	18/03/2017	20/03/2017	21/03/2017
14	18/03/2017	20/02/2017	20/02/2017	05/02/2017	21/03/2017	08/03/2017
15	20/03/2017	22/02/2017	22/02/2017	10/02/2017	22/03/2017	09/03/2017
16	22/03/2017	14/04/2017	15/04/2017	15/02/2017	23/03/2017	10/03/2017
17	24/03/2017	26/02/2017	06/06/2017	20/02/2017	24/03/2017	11/03/2017
18	26/03/2017	28/02/2017	28/07/2017	25/02/2017	25/03/2017	12/03/2017

Figure 7-14 – *Les cellules dont la date est supérieure à l'échéance indiquée par l'utilisateur sont mises en évidence (cadre gris clair).*

Attention

Si l'utilisateur clique sur le bouton Annuler ou sur le bouton de fermeture d'une boîte de dialogue affichée à l'aide de `InputBox`, la fonction renvoie une chaîne vide. Une erreur pourra alors être générée par le programme s'il tente de l'exploiter. Placez une instruction `If…Then…Else` pour vérifier que la valeur retournée par `InputBox` n'est pas une chaîne vide. Les instructions suivantes pourront être placées sous la ligne 11 du programme précédent, afin de mettre fin à la procédure si l'utilisateur annule la saisie d'une valeur.

```
If DateEcheance="" Then
  Exit Sub
End if
```

De la même façon, si l'utilisateur saisit n'importe quoi d'autre que la valeur attendue, le programme pourra générer une erreur ou produire des résultats erronés. Utilisez les instructions de contrôle de type de données présentées au chapitres 6 pour vous assurer que les informations fournies sont valides. Vous apprendrez à gérer ces éventuelles erreurs au chapitres 10.

La méthode InputBox

L'objet `Application` d'Excel possède une méthode `InputBox`, que vous pouvez substituer à la fonction éponyme de Visual Basic. L'intérêt est qu'elle permet de spécifier le type de données qui sera renvoyé. Utilisez cette méthode selon la syntaxe suivante :

```
Application.InputBox(prompt, title, default, left, top, helpFile, helpContextID, type)
```

Si vous ne spécifiez pas de valeur pour l'argument *`title`*, le titre par défaut de la boîte de dialogue sera « Entrée ». Les arguments nommés *`left`* et *`top`* spécifient l'emplacement de la boîte de dialogue sur l'écran au moment de son affichage, tandis que *`helpFile`* et *`helpContextID`* servent à associer des fichiers d'aide à la boîte de dialogue. Ils existent aussi pour la fonction `InputBox` de Visual Basic.

L'argument de type `Variant` *`type`* est facultatif. Il peut prendre l'une des valeurs présentées dans le tableau 7-2 et détermine le type de données renvoyé. La méthode `InputBox` peut renvoyer plusieurs types de données définis ; dans ce cas, affectez à *`type`* la somme des valeurs correspondantes (tableau 7-2). Par exemple, pour que l'utilisateur soit autorisé à indiquer un nombre ou une référence de cellules, vous lui affecterez la valeur 9 (1 + 8).

Tableau 7-2. Valeurs admises par l'argument type de la méthode InputBox d'Excel

Valeur de Type	Type de données renvoyé par InputBox
0	Formule
1	Valeur numérique
2	Chaîne de caractères
4	Valeur booléenne (`False` ou `True`)
8	Référence de cellule (objet `Range`)
16	Valeur d'erreur
64	Tableau de valeurs

Conseil

Si l'information saisie par l'utilisateur dans la zone de texte ne correspond pas au type de données déclaré pour la méthode `InputBox`, une erreur sera générée. Pensez à créer un gestionnaire d'erreurs (voir chapitres 10).

L'autre avantage de la méthode `InputBox` d'Excel sur sa concurrente Visual Basic est de permettre à l'utilisateur de sélectionner une plage de cellules avant de cliquer sur le bouton OK. Il est ainsi possible de sélectionner un classeur ou

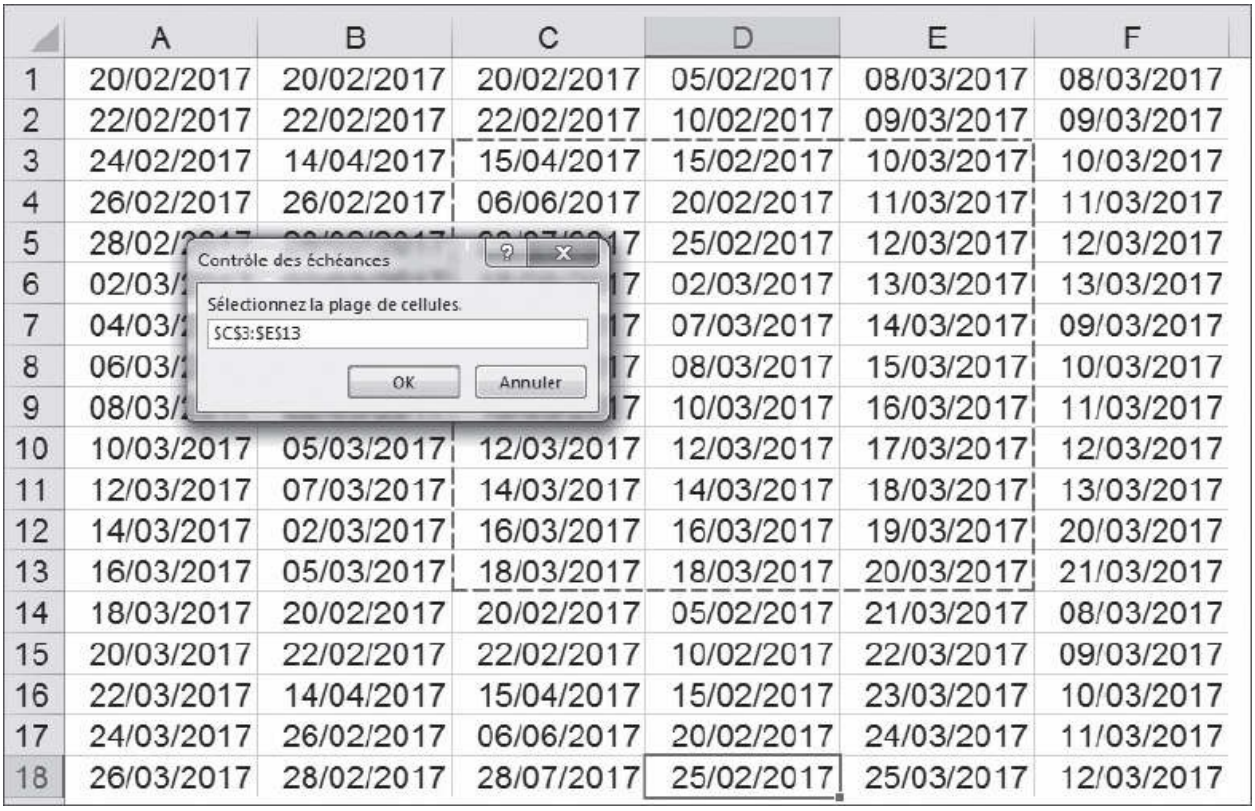
une feuille spécifique, puis de sélectionner la plage voulue. Les coordonnées de celles-ci s'affichent alors dans la zone de texte de la boîte de dialogue. L'instruction suivante affiche une boîte de dialogue qui accepte pour valeur une référence de cellule. L'utilisateur est invité à sélectionner une plage de cellules dans la feuille active (voir figure 7-15).

```
Sub RenvoyerUnePlageAvecInputBox()
  Dim MaPlage As Range
  Set MaPlage = Application.InputBox(prompt:="Sélectionnez la plage de cellules.", _
    Title:="Contrôle des échéances", Left:=3, Top:=-80, Type:=8)
End Sub
```

Entraînez-vous ! Modifiez la procédure `VerifierEcheances` de façon à inviter l'utilisateur à sélectionner la plage de cellules à traiter, plutôt que de traiter la plage sélectionnée au moment de l'exécution de la macro.

Conseil

Lorsque vous utilisez la fonction `InputBox` d'Excel pour inviter l'utilisateur à sélectionner une plage de cellules, tirez profit des arguments `Left` et `Top`. Si la boîte de dialogue s'affiche dans l'angle supérieur gauche de la fenêtre, l'utilisateur n'aura pas besoin de la déplacer pour sélectionner des cellules sur la feuille.



	A	B	C	D	E	F
1	20/02/2017	20/02/2017	20/02/2017	05/02/2017	08/03/2017	08/03/2017
2	22/02/2017	22/02/2017	22/02/2017	10/02/2017	09/03/2017	09/03/2017
3	24/02/2017	14/04/2017	15/04/2017	15/02/2017	10/03/2017	10/03/2017
4	26/02/2017	26/02/2017	06/06/2017	20/02/2017	11/03/2017	11/03/2017
5	28/02/			25/02/2017	12/03/2017	12/03/2017
6	02/03/			02/03/2017	13/03/2017	13/03/2017
7	04/03/			07/03/2017	14/03/2017	09/03/2017
8	06/03/			08/03/2017	15/03/2017	10/03/2017
9	08/03/			10/03/2017	16/03/2017	11/03/2017
10	10/03/2017	05/03/2017	12/03/2017	12/03/2017	17/03/2017	12/03/2017
11	12/03/2017	07/03/2017	14/03/2017	14/03/2017	18/03/2017	13/03/2017
12	14/03/2017	02/03/2017	16/03/2017	16/03/2017	19/03/2017	20/03/2017
13	16/03/2017	05/03/2017	18/03/2017	18/03/2017	20/03/2017	21/03/2017
14	18/03/2017	20/02/2017	20/02/2017	05/02/2017	21/03/2017	08/03/2017
15	20/03/2017	22/02/2017	22/02/2017	10/02/2017	22/03/2017	09/03/2017
16	22/03/2017	14/04/2017	15/04/2017	15/02/2017	23/03/2017	10/03/2017
17	24/03/2017	26/02/2017	06/06/2017	20/02/2017	24/03/2017	11/03/2017
18	26/03/2017	28/02/2017	28/07/2017	25/02/2017	25/03/2017	12/03/2017

Figure 7-15 – *La zone de texte de la boîte de dialogue reflète la sélection effectuée sur la feuille Excel.*

La fonction MsgBox

La fonction `MsgBox` affiche une boîte de dialogue présentant un message et des boutons de commande, pour donner une information à l'utilisateur ou obtenir une réponse à une question qui orientera l'exécution du programme. Une valeur de type `Integer` est renvoyée en fonction du bouton sur lequel l'utilisateur a cliqué, et stockée dans une variable pour être ensuite exploitée par le programme.

La fonction `MsgBox` s'utilise selon la syntaxe suivante :

```
Variable = MsgBox(prompt, buttons, title)
```

`prompt` est un argument nommé de type `String`, correspondant au message affiché dans la boîte de dialogue. *`buttons`* est un argument nommé de type numérique facultatif. Il détermine les boutons affichés dans la boîte de dialogue, le symbole identifiant le type du message (information, question, etc.) et le bouton par défaut. Si cet argument est omis, un seul bouton libellé OK s'affiche et aucune icône n'identifie le type du message. *`title`* est un argument nommé de type `String` facultatif, affiché dans la barre de titre de la boîte de dialogue. Si cet argument est omis, c'est le nom de l'application qui apparaît.

L'argument *`buttons`* est défini par la somme des valeurs choisies pour chacun des groupes présentés dans le tableau 7-3. Votre code gagnera cependant en lisibilité si vous utilisez les constantes VBA intégrées plutôt que des valeurs numériques.

Tableau 7-3. Définition de l'argument buttons

Constante	Valeur	Description
Bouton		
`vbOKOnly`	0	OK
`vbOKCancel`	1	OK et Annuler
`vbAbortRetryIgnore`	2	Abandonner, Réessayer et Ignorer
`vbYesNoCancel`	3	Oui, Non et Annuler
`vbYesNo`	4	Oui et Non
`vbRetryCancel`	5	Réessayer et Annuler
Symbole		
`vbCritical`	16	Message critique
`vbQuestion`	32	Question
`vbExclamation`	48	Stop
`vbInformation`	64	Information

Bouton actif par défaut		
`vbDefaultButton1`	0	Premier bouton
`vbDefaultButton2`	256	Deuxième bouton
`vbDefaultButton3`	512	Troisième bouton
`vbDefaultButton4`	768	Quatrième bouton

Si vous souhaitez, par exemple, afficher une boîte de dialogue contenant le symbole « Question », dans laquelle l'utilisateur pourra répondre par « Oui » ou « Non », l'argument *`buttons`* sera égal à 36 : 4 (pour les boutons) + 32 (pour le symbole). Les trois instructions suivantes sont équivalentes et entraînent l'affichage de la boîte de dialogue représentée à la figure 7-16. La troisième formule est cependant plus compréhensible.

```
réponse = MsgBox("Autre recherche ?", 36, "Recherche")
réponse = MsgBox("Autre recherche ?", 32 + 4, "Recherche")
réponse = MsgBox("Autre recherche ?", vbYesNo + vbQuestion, "Recherche")
```

Figure 7-16 – *Une boîte de dialogue affichée à l'aide de l'instruction MsgBox.*

Astuce

Lorsque vous insérez la fonction `MsgBox` dans une procédure, placez le curseur sur la fonction, puis tapez sur la touche F1 afin d'en activer la rubrique d'aide. Vous pourrez ainsi consulter le tableau répertoriant les constantes à attacher à l'argument `buttons`.

Si vous souhaitez afficher une boîte de dialogue dans le seul objectif de transmettre une information à l'utilisateur (ne comportant qu'un bouton OK), il est inutile d'affecter une variable à la fonction `MsgBox`. Cependant, une fonction ne peut être utilisée que dans une instruction d'affectation. Vous devez alors utiliser l'instruction `MsgBox`. Sa syntaxe est la même, à la différence près que ses arguments ne sont pas encadrés par des parenthèses. L'instruction suivante affiche la boîte de dialogue présentée à la figure 7-17.

```
MsgBox "Les documents sont en cours d'impression.", vbInformation
```

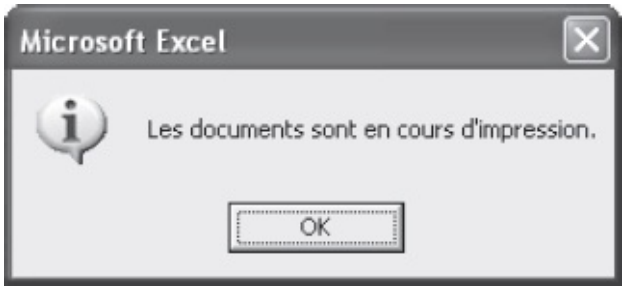


Figure 7-17 – *Pour afficher un simple message informatif, utilisez une instruction plutôt qu'une fonction.*

Par défaut, c'est le premier bouton qui est actif (celui qui sera validé si l'utilisateur appuie sur la touche Entrée). Si une réponse positive de la part de l'utilisateur peut être lourde de conséquences, il vaut mieux définir le deuxième ou le troisième comme bouton par défaut. L'instruction suivante affiche la boîte de dialogue présentée à la figure 7-18 :

```
réponse = MsgBox("Supprimer toutes les informations ?", _
  vbYesNoCancel + vbCritical + vbDefaultButton2)
```

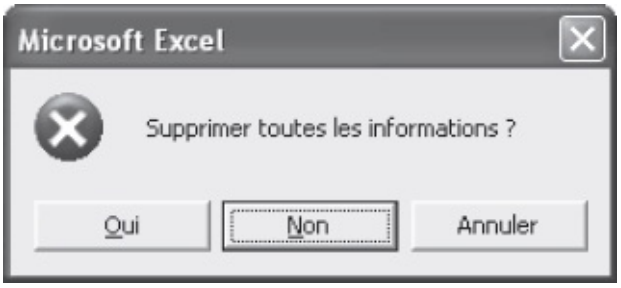


Figure 7-18 – *Si l'utilisateur appuie sur la touche Entrée du clavier, c'est la réponse « Non » qui sera validée.*

La valeur renvoyée par la fonction `MsgBox` varie en fonction du bouton sur lequel l'utilisateur a cliqué, selon la règle suivante :

Bouton	Constante	Valeur
OK	`vbOK`	1
Annuler	`vbCancel`	2
Abandonner	`vbAbort`	3
Réessayer	`vbRetry`	4
Ignorer	`vbIgnore`	5
Oui	`vbYes`	6
Non	`vbNo`	7

Une instruction conditionnelle est utilisée pour tester la variable recevant l'information et orienter le déroulement du programme en conséquence. Là encore, vous pouvez procéder en utilisant la constante VBA ou la valeur numérique correspondante. Dans le cas de la boîte de dialogue présentée à la figure 7-18, si l'utilisateur clique sur le bouton Oui, la valeur 6 (correspondant à la constante `vbYes`) sera affectée à la variable réponse. S'il clique sur le bouton Non, la valeur 7 (`vbNo`) sera retournée. Enfin, s'il choisit le bouton Annuler, c'est la valeur 2 (`vbCancel`) qui sera renvoyée.

Le programme suivant reprend l'exemple donné précédemment pour `InputBox`. Une fonction `MsgBox` a été utilisée afin de proposer à l'utilisateur de définir une autre date, en cas de dépassement d'échéance. S'il n'y a pas de dépassement d'échéance, une autre boîte de dialogue sera affichée pour en informer l'utilisateur.

```
 1: Sub VerifierEcheances()
 2:   Dim ZoneATester As String
 3:   ZoneATester = ActiveWindow.RangeSelection.Address
 4:   If ZoneATester=Null Then
 5:     MsgBox "Sélectionnez la plage de cellules à tester.", vbOKOnly + vbInformation
 6:     Exit Sub
 7:   End If
 8:   Dim DateEcheance As Variant
 9:   DateEcheance = InputBox("Indiquez la date d'échéance.", _
       "Echéance des opérations en cours", Date + 30)
10:   DateEcheance = CDate(DateEcheance)


11:   Dim CellTest As Range
12:   Dim DatesHorsEcheance As Boolean
13:   DatesHorsEcheance = False
14:   For Each CellTest In Range(ZoneATester)
15:     If IsDate(CellTest)=True Then
16:       If CellTest.Value>DateEcheance Then
17:         CellTest.Interior.ColorIndex = 6
18:         DatesHorsEcheance = True
19:       End If
20:     End If
21:   Next
22:   If DatesHorsEcheance=True Then
23:     Dim RedéfinirLaDate As Integer
24:     RedéfinirLaDate = MsgBox("Des problèmes d'échéance ont été trouvés." & _
         Chr(10) & "Souhaitez-vous spécifier une autre date ?", _
         vbYesNo + vbQuestion, "Redéfinir la date d'échéance ?")
25:     If RedéfinirLaDate=vbYes Then
26:       Range(ZoneATester).Interior.ColorIndex = xlNone
27:       Call VerifierEcheances
28:     End If
29:   Else
29:     MsgBox "Pas de problème d'échéance.", _
       vbOKOnly + vbInformation, "Echéancier respecté"
30:   End If
31: End Sub
```

Lignes 12 et 13, la variable `DatesHorsEcheance` – de type `Boolean` – est déclarée et se

voit affecter la valeur `False`. Ligne 18, une instruction lui affecte la valeur `True` si une date supérieure à celle spécifiée par l'utilisateur est trouvée.

La mise en valeur des éventuelles cellules hors échéance terminée, une instruction conditionnelle `If…Then…Else` est utilisée pour afficher un message (lignes 22 à 30). Si `DateHorsEcheance` renvoie `True` (des cellules contenant des dates au-delà de celle spécifiée par l'utilisateur sont trouvées), les instructions des lignes 23 à 28 sont exécutées. La boîte de dialogue représentée à la figure 7-19 est alors affichée. La valeur renvoyée par la fonction `MsgBox` est stockée dans la variable `RedéfinirLaDate`. Une structure conditionnelle imbriquée en teste ensuite la valeur. Si elle renvoie `vbYes` (l'utilisateur a cliqué sur le bouton Oui), la couleur d'intérieur des cellules de la zone sélectionnée est supprimée (ligne 26) et la fonction s'appelle elle-même (ligne 27). Si `DateHorsEcheance` renvoie `False`, une boîte de dialogue s'affiche (ligne 29) afin d'informer l'utilisateur qu'il n'y a pas de problème.

Définition

Une procédure qui s'appelle elle-même est dite récursive.

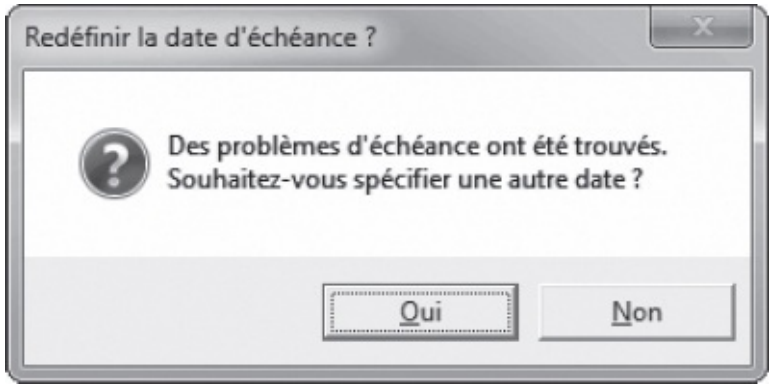


Figure 7-19 – *L'utilisateur peut redéfinir une date d'échéance.*

Affichage de boîtes de dialogue Excel

Il peut être utile d'afficher des boîtes de dialogue Excel à un moment spécifique de l'exécution d'un programme, si l'on veut par exemple que l'utilisateur puisse indiquer le dossier d'enregistrement d'un classeur.

Les boîtes de dialogue prédéfinies d'Excel

Les boîtes de dialogue sont des objets `Dialog`. Pour en afficher une, faites appel

à la collection `Dialogs` et appliquez la méthode `Show` ou `Display` à l'objet défini selon la syntaxe suivante :

```
Application.Dialogs(xlDialog).Show
Application.Dialogs(xlDialog).Display
```

`xlDialog` est une constante Excel qui indique la boîte à afficher.

Par exemple, l'instruction suivante affiche la boîte de dialogue Ouvrir :

```
Application.Dialogs(xlDialogOpen).Show
```

Lorsqu'une boîte de dialogue Excel s'affiche à l'aide de la méthode `Show`, les commandes qu'elle prend en charge s'exécutent normalement si l'utilisateur valide les paramètres qu'il y a définis. En revanche, la méthode `Display` affiche la boîte de dialogue et permet de récupérer les informations au moment de la validation par l'utilisateur, sans en exécuter les commandes. La procédure se poursuit ensuite avec l'instruction située derrière celle qui a appelé la boîte de dialogue.

Vous pouvez évidemment affecter un objet `Dialog` à une variable de type `Object` ou `Variant` et faire appel à cet objet pour afficher la boîte en question. Les instructions suivantes affichent la boîte de dialogue Enregistrer sous.

```
Dim BoîteEnregistrerSous as Dialog
Set BoîteEnregistrerSous = Application.Dialogs(xlDialogSaveAs)
BoîteEnregistrerSous.Show
```

Vous trouverez une liste des constantes `xlDialog` dans l'aide de VBA pour Excel. Référez-vous à la rubrique Références pour Microsoft Excel/Objets/Dialogs.

Les méthodes GetOpenFilename et GetSaveAsFilename

Les méthodes `GetOpenFilename` et `GetSaveAsFilename` d'Excel affichent les boîtes de dialogue standards Ouvrir et Enregistrer sous afin de récupérer le nom d'un fichier. Celles-ci sont plus souples et plus fonctionnelles que les objets `Dialogs` correspondants.

Attention

Contrairement à ce qui se passe lorsque vous appliquez la méthode `Show` à un objet `Dialog`, quand une boîte de dialogue s'affiche à l'aide de `GetOpenFilename` ou `GetSaveAsFilename`, aucune action n'est exécutée lorsque l'utilisateur clique sur le bouton de validation. Ces méthodes permettent simplement de récupérer un nom de fichier. Ce fichier doit ensuite être manipulé par programmation.

Utilisez les méthodes `GetOpenFilename` et `GetSaveAsFilename` selon la syntaxe suivante :

```
Application.GetSaveAsFilename(InitialFilename, FileFilter, FilterIndex, Title,
➥ ButtonText)
```

```
expression.GetOpenFilename(FileFilter, FilterIndex, Title, ButtonText, MultiSelect)
```

Tableau 7-4. Arguments des méthodes GetOpenFilename et GetSaveAsFilename

Argument	Description
`InitialFilename`	Argument facultatif de type `Variant`. Nom de fichier par défaut apparaissant dans la zone de texte Nom. Si cet argument est omis, le nom du classeur actif est utilisé par défaut.
`FileFilter`	Argument facultatif de type `Variant`. Critères de filtrage des fichiers. Chaque critère doit apparaître sous la forme d'une paire composée du filtre de fichier, suivi de la spécification de son extension, tels qu'ils apparaissent dans la zone Type de fichier. Les paires sont également séparées par des virgules. Si plusieurs extensions sont associées à un type de fichier, elles doivent être séparées par un point-virgule. Par exemple, la valeur suivante : `"Classeur Microsoft Excel (*.xlsx),*.xlsx,PageWeb (*.htm; *.html), *.htm;*.html"` définie pour l'argument `FileFilter` laissera apparaître deux types de fichiers.
`FilterIndex`	Argument facultatif de type `Variant`. Index du critère de filtrage par défaut : 1 pour le premier, 2 pour le deuxième, etc. Le premier filtre de fichier est utilisé si l'argument n'a pas été spécifié ou s'il est supérieur au nombre de filtres disponibles.
`Title`	Argument facultatif de type `Variant`. Texte apparaissant dans la barre de titre de la boîte de dialogue. Si cet argument est omis, le titre de la boîte de dialogue est Ouvrir.
`ButtonText`	Argument facultatif de type `Variant`. Sur Macintosh uniquement. Le libellé du bouton Ouvrir.
`MultiSelect`	Argument facultatif de type `Variant`. Si `MultiSelect` a la valeur `True`, l'utilisateur peut effectuer une sélection multiple. Si `MultiSelect` a la valeur `False` (valeur par défaut), l'utilisateur ne peut sélectionner qu'un fichier. Si plusieurs fichiers sont sélectionnés, la valeur est renvoyée sous forme de tableau et doit donc être stockée dans une variable de type `Variant` ou `Array`.

Utilisez l'instruction `ChDir` pour modifier le dossier proposé par défaut dans la boîte de dialogue affichée.

L'exemple suivant affiche la boîte de dialogue représentée à la figure 7-20, puis une boîte indiquant la liste des fichiers choisis par l'utilisateur (voir figure 7-21).

```
1: Sub OuvertureDeFichiers()
2:    Dim OuvrirFichiers As Variant
3:    'modification du chemin par défaut
4:    ChDir ("C:\Users\Nom_utilisateur\Desktop\")
5:    'affichage de la boîte de dialogue Ouvrir
6:    OuvrirFichiers = Application.GetOpenFilename(filefilter:="Classeur Microsoft
        ➥ Excel (*.xlsx),*.xlsx,PageWeb (*.htm; *.html),*.htm;*.html",
        ➥ filterindex:=2, Title:="Ouverture des fichiers ventes", MultiSelect:=True)
7:    'si l'utilisateur a sélectionné plusieurs fichiers
8:    If UBound(OuvrirFichiers)>1 Then
9:       Dim rep As Long
```

```
10:       Dim Liste As String
11:       Dim compteur As Byte
12:       For compteur = 1 To UBound(OuvrirFichiers)
13:          Liste = Liste & vbCr & OuvrirFichiers(compteur)
14:       Next compteur
15:       'affichage de l'ensemble de la liste des fichiers et proposition d'ouverture
16:       rep = MsgBox("L'utilisateur a sélectionné plusieurs fichiers. En voici la
            ➥ liste." & _
17:          Liste & vbCr & "Voulez-vous les ouvrir ?", vbYesNo + vbQuestion, _
18:          "Ouvrir les fichiers ?")
19:
20:       'ouverture des fichiers en cas de réponse positive
21:       If rep=vbYes Then
22:          For compteur = 1 To UBound(OuvrirFichiers)
23:             Workbooks.Open Filename:=OuvrirFichiers(compteur)
24:          Next compteur
25:       End If
26:    'si un seul fichier a été sélectionné, il est ouvert
27:    Else
28:       Workbooks.Open Filename:=OuvrirFichiers(1)
29:    End If
30: End Sub
```

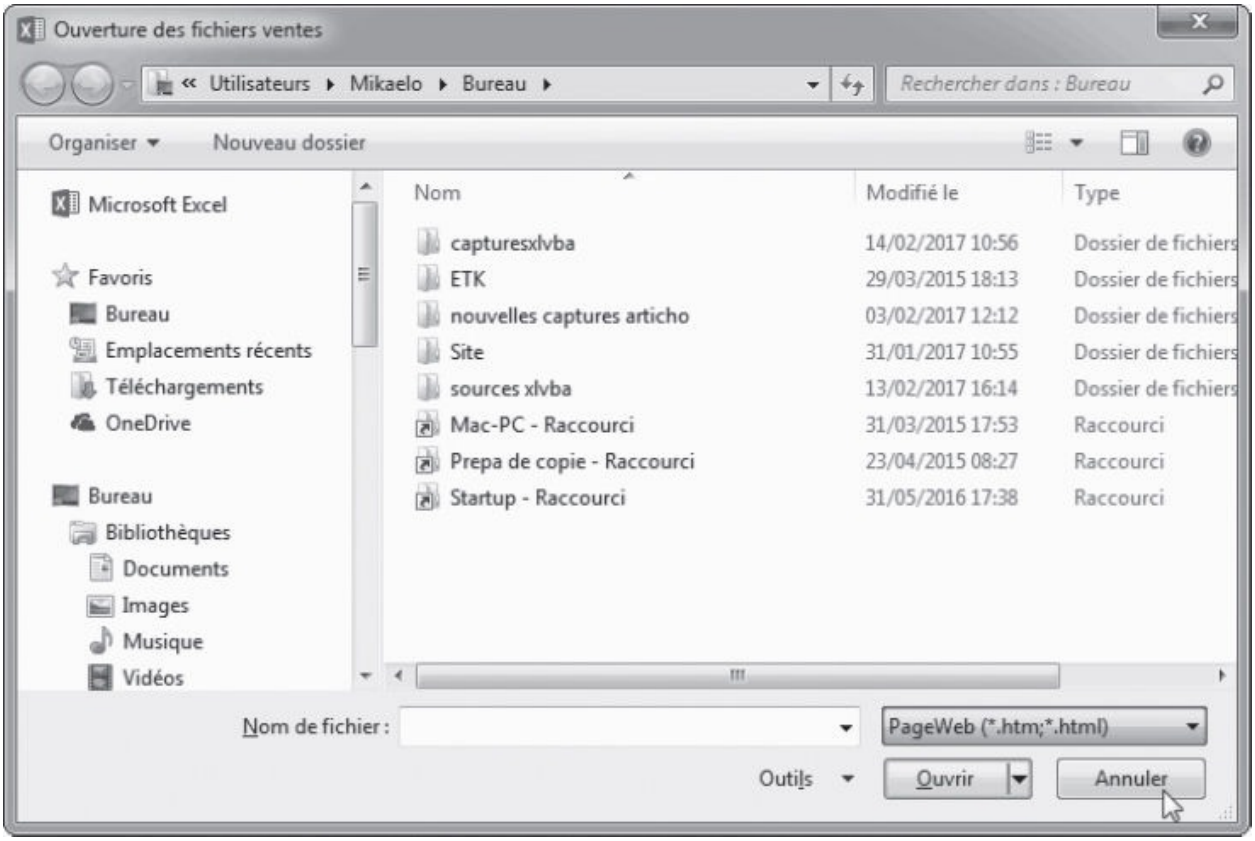


Figure 7-20 – *La méthode GetOpenFileName affiche une boîte de dialogue Ouvrir personnalisée.*

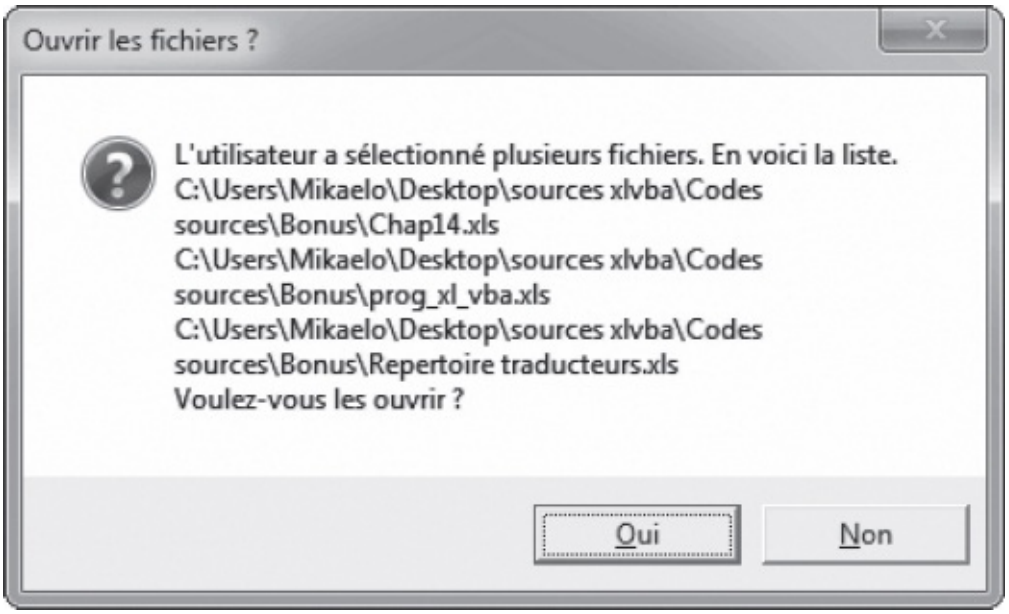


Figure 7-21 – *Les fichiers sélectionnés sont récupérés dans une variable de matrice.*

La variable `OuvrirFichiers` déclarée ligne 2 servira à stocker le(s) fichier(s) sélectionné(s) par l'utilisateur. Ligne 4, le chemin par défaut est modifié à l'aide de l'instruction `ChDir`, pour que le Bureau Windows soit proposé.

Ligne 6, la boîte de dialogue Ouvrir s'affiche. Le résultat de la sélection opérée par l'utilisateur est affecté à la variable `OuvrirFichiers`. L'argument `filefilter` est défini pour que seuls les fichiers portant l'extension .xlsx, .htm ou .html soient proposés. L'argument `filterindex` définit le deuxième type de fichier (.htm ou .html) comme type par défaut. Enfin, le titre de la boîte de dialogue est défini à Ouverture des fichiers ventes, et la sélection multiple est autorisée.

Ligne 8, une instruction conditionnelle `If…Then…Else` vérifie si plusieurs fichiers ont été sélectionnés. Si tel est le cas, les instructions des lignes 8 à 26 s'exécutent.

Lignes 9 à 11, les variables `rep`, `Liste` et `Compteur` sont déclarées. Lignes 12 à 14, une instruction `For…Each…Next` stocke la liste des fichiers sélectionnés dans la variable `Liste`. La valeur initiale du compteur est 1 ; elle est incrémentée de 1 à chaque passage de la boucle jusqu'à atteindre la valeur limite de la variable de matrice [`UBound(OuvrirFichiers)`]. À chaque passage, la variable `Liste` reçoit sa valeur + un retour chariot (`& vbCr`) + la valeur stockée dans la variable `OuvrirFichiers`, à la position correspondant à la valeur de `Compteur` [`OuvrirFichiers(compteur)`]. Lignes 16 à 18, une instruction `MsgBox` est utilisée pour afficher la liste des fichiers sélectionnés par l'utilisateur et lui proposer de les

ouvrir (voir figure 7-21). Lignes 21 à 25, une instruction conditionnelle imbriquée évalue la réponse de l'utilisateur. Une instruction `For…Next` ouvre les fichiers en cas de réponse positive. Si un seul fichier a été sélectionné (ligne 27), il est ouvert (ligne 28).

> **Info**
>
> Si l'un des classeurs sélectionnés ne s'affiche pas, c'est qu'il est masqué. Utilisez la commande Afficher de l'onglet Affichage du ruban.

Tel qu'il se présente ici, ce programme générera une erreur si l'utilisateur clique sur le bouton Annuler ou sur la croix de fermeture de la boîte de dialogue. Ajoutez les instructions qui suivent entre les lignes 6 et 7 du programme précédent pour résoudre ce problème. Si aucun fichier n'est sélectionné, le message de la figure 7-22 s'affiche et la procédure prend fin.

```
If OuvrirFichiers=False Then
   MsgBox "Aucun fichier n'a été sélectionné. Fin de la procédure", _
      VbOKOnly + vbCritical, "Fin de la procédure"
   Exit Sub
End If
```

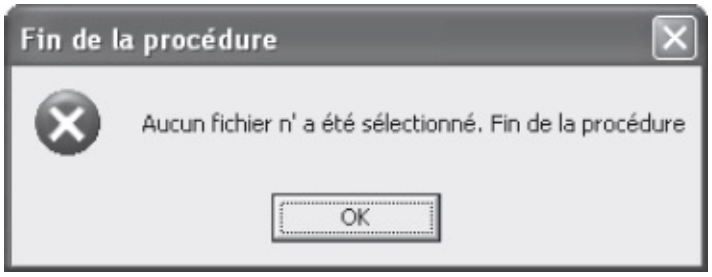


Figure 7-22 – *L'annulation de l'ouverture est maintenant gérée par le programme.*

Utiliser les opérateurs logiques

Visual Basic propose des opérateurs logiques, qui aideront à optimiser les instructions conditionnelles `While…Wend` et `If…Then…Else`. Ils combinent en effet plusieurs conditions et simplifient les instructions conditionnelles dans bien des cas. L'instruction suivante utilise l'opérateur logique `And` pour vérifier que la valeur de la cellule A3 est inférieure à 1 000 et supérieure à 2 000 :

```
If Range("A3").Value<1000 and Range("A3").Value>2000 Then…
```

Les conditions établies à l'aide d'opérateurs logiques sont proches d'une condition exprimée dans un langage courant, ce qui en facilite

considérablement la compréhension.

Les instructions disponibles pour utiliser les opérateurs logiques sont présentées dans le tableau 7-5.

Tableau 7-5. Les opérateurs logiques

Opérateur	Description
`Or`	Contrôle que l'une des conditions spécifiées est vérifiée. Si c'est le cas, la condition renvoie `True`.
`And`	Contrôle que toutes les conditions spécifiées sont vérifiées. Si c'est le cas, l'expression conditionnelle renvoie `True`.
`Xor`	Ou restrictif. Si l'une ou l'autre des conditions est respectée, l'expression conditionnelle renvoie `True`. Si plus d'une condition est vérifiée, elle renvoie `False`.
`Not`	Cet opérateur nie l'expression devant laquelle il est situé. La structure conditionnelle `If Not condition Then…` est vérifiée si la *`condition`* n'est pas respectée.

Astuce

Vous pouvez employer autant d'opérateurs `Or` et `And` que vous voulez dans une même expression. Cependant, pour les combiner, vous devez utiliser des parenthèses afin de spécifier quels sont les rapports entretenus entre les différentes conditions. Vous obtiendrez alors des expressions du type :

```
If (Condition1 Or Condition2) And Condition3 Then
  Série d'instructions
End If
```

Dans le cas d'une telle expression, la Série d'instructions ne sera exécutée que si l'une des deux premières conditions (ou les deux) est vérifiée et si la Condition3 est aussi vérifiée.

Trier des données

Trier et rechercher des données est une opération courante pour l'utilisateur d'Excel. La recherche étant en général subordonnée au tri des données, cette question est fondamentale pour le développeur. Un programme VBA peut trier une zone définie d'une feuille Excel comme un utilisateur le ferait *via* le bouton Trier de l'onglet Données, en faisant appel à la méthode `Sort`.

Vous serez cependant amené à trier des données stockées dans une variable et non sur une feuille Excel. Par ailleurs, si un programme effectue de nombreuses opérations d'analyse, de traitement et de tri sur des données, vous gagnerez en rapidité d'exécution en travaillant sur ces données en mémoire plutôt qu'à même la feuille Excel. La méthode la plus simple consiste alors à stocker les données de la feuille dans une variable tableau et à travailler sur cette variable.

Les techniques de tri des tableaux sont nombreuses. Nous vous proposons ici une méthode simple et efficace, qui consiste à comparer des éléments adjacents et à les intervertir si nécessaire. On utilise pour ce faire une boucle `For…Next` imbriquée. Considérez la fonction suivante :

```
 1: Public Function TriTableau(MaVar())
 2:  Dim i As Long
 3:  Dim j As Long
 4:  Dim temp
 5:  For i = LBound(MaVar) To UBound(MaVar) - 1
 6:    For j = i + 1 To UBound(MaVar)
 7:      If MaVar(i)>MaVar(j) Then
 8:        temp = MaVar(i)
 9:        MaVar(i) = MaVar(j)
10:        MaVar(j) = MaVar(i)
11:      End If
12:    Next j
13:  Next i
14:  TriTableau = MaVar
15: End Function
```

La déclaration de notre fonction indique qu'elle attend la variable de tableau `MaVar` en argument. Lignes 2 à 4, les variables sont déclarées. `i` et `j` sont les compteurs de nos boucles `For…Next`, tandis que la variable `temp` servira simplement à stocker provisoirement des données.

Lignes 5 à 13 se trouvent les boucles `For…Next` imbriquées, qui assurent le tri de notre tableau. La boucle principale s'exécute autant de fois que la variable contient d'espaces de stockage – 1. Elle trie la variable indice par indice, du premier au dernier.

La boucle imbriquée (lignes 6 à 12) effectue le tri à proprement parler. La valeur traitée est comparée à toutes les valeurs qui sont stockées après elle dans le tableau (`j=i+1 To UBound(MaVar)`). Si la valeur traitée est supérieure à celle à laquelle elle est comparée (ligne 7), les deux sont interverties (lignes 8 à 10). On stocke pour cela la valeur de `mavar(i)` dans la variable `temp`, puis on lui substitue la valeur de `mavar(j)`. On remplace ensuite la valeur de `mavar(j)` par la valeur de `mavar(i)` précédemment mémorisée dans la variable `temp`.

Une fois la boucle imbriquée exécutée, la valeur stockée à l'indice i de notre variable est plus petite que toutes celles qui la suivent. La structure `For…Next` principale reprend alors la main et passe à l'indice suivant de la variable tableau. Une fois l'opération menée pour l'ensemble des indices de la variable, les valeurs sont stockées par ordre croissant. Notre fonction reçoit alors la valeur de `MaVar` (ligne 14) et prend fin, retournant la variable maintenant triée.

Astuce

Pour un tri décroissant, remplacez simplement le signe > de la ligne 7 par le signe <.

Dans l'exemple suivant, on utilise la même méthode de comparaison pour trier les feuilles de travail du classeur actif. Cette fois-ci, les données ne sont pas stockées dans un tableau, mais les feuilles sont directement triées sur le classeur.

```
 1: Sub TriFeuilles()
 2:   Dim I As Integer
 3:   Dim J As Integer
 4:   Dim Min As Integer
 5:   Dim ModeCalcul As Integer
 6:   ModeCalcul = Application.Calculation
 7:   Application.Calculation = xlCalculationManual
 8:   Application.ScreenUpdating = False
 9:   With ActiveWorkbook.Worksheets
10:     For I = 1 To .Count - 1
11:       Min = I
12:       For J = I + 1 To .Count
13:         If .Item(J).Name<.Item(Min).Name Then Min = J
14:       Next J
15:       If Min<>I Then .Item(Min).Move before:=Worksheets(I)
16:     Next I
17:   End With
18:   Application.Calculation = ModeCalcul
19:   Application.ScreenUpdating = True
20: End Sub
```

Lignes 2 à 5, les variables sont déclarées. Ligne 6, la valeur `Application.Calculation` (le mode de calcul défini pour le classeur) est mémorisée. Les lignes 7 et 8 sont destinées à optimiser le temps d'exécution de la macro. Le classeur est ainsi passé en mode de calcul manuel ligne 7, afin d'empêcher un éventuel recalcul des valeurs des cellules à chaque déplacement d'une feuille. Ligne 8, la valeur `False` est affectée à la propriété `ScreenUpdating` de l'objet `Application`, afin de ne pas mettre à jour l'affichage à l'écran lors de l'exécution de la macro.

Lignes 9 à 17, une structure `With…End With` est utilisée avec l'objet `ActiveWorkbook.Worksheets` pour simplifier l'écriture du code. Lignes 10 à 16, deux boucles `For…Next` imbriquées assurent le tri des feuilles.

La première boucle s'exécute autant de fois qu'il y a de feuilles dans le classeur – 1. À chaque passage, la variable `Min` reçoit pour valeur l'index de la feuille en cours de traitement. La boucle imbriquée (lignes 12 à 14) compare alors le nom stocké dans `Min` avec celui de chaque feuille située après elle. Si un nom inférieur (alphabétiquement antérieur) est trouvé, `Min` reçoit pour valeur l'index de la feuille portant ce nom.

À la fin de cette boucle, `Min` contient donc la valeur d'index de la feuille portant le nom le plus petit parmi les valeurs qui ont été comparées. Ligne 15, on

vérifie si on a trouvé une feuille portant un nom plus petit lors des comparaisons (`If Min<>I`). Si c'est le cas, cette feuille est placée devant la feuille en cours de traitement (`.Item(Min).Move before:=Worksheets(I)`).

La boucle principale reprend alors la main, et la feuille suivante est traitée selon le même principe. Une fois les feuilles du classeur triées par ordre croissant de nom, le mode de calcul du classeur est redéfini à son état initial (ligne 18), tandis que la mise à jour de l'affichage écran est réactivée (ligne 19).

Conseil

Notez que, dans l'exemple précédent, on évite des opérations inutiles de déplacement de feuilles, en mémorisant la donnée la plus petite lors des comparaisons (ligne 13) et en n'effectuant qu'une seule substitution de position si nécessaire à la fin du passage de la boucle imbriquée (ligne 15). Exercez-vous à modifier la fonction `TriTableau` présentée plus avant selon le même principe.

8

Fonctions Excel et VBA

Si les fonctions sont un élément clé du développement d'applications, quels que soient le langage et l'environnement de programmation, c'est particulièrement vrai pour Excel, dont l'activité principale consiste à effectuer des calculs.

Vos projets exploiteront aussi bien des fonctions d'Excel que de VBA, mais également celles que vous allez créer.

Utiliser les fonctions Excel dans VBA

Excel offre un nombre impressionnant de fonctions intégrées, de la simple addition aux calculs avancés. Pour la plupart, elles peuvent être manipulées *via* du code Visual Basic.

> **Attention**
>
> Ne confondez pas les fonctions spécifiques à Excel et celles de Visual Basic. Si de nombreuses fonctions mathématiques Excel ont leurs équivalents Visual Basic, le tableur en propose dans le domaine des statistiques, de la finance, etc., qui lui sont propres et n'existent pas en Visual Basic. Un programme exploitant ces fonctions ne pourra donc pas être exécuté dans une application hôte autre qu'Excel.

Pour utiliser une fonction Excel dans un programme VBA, vous ferez appel à l'objet `Application` représentant l'application hôte (en l'occurrence Excel) qui la contient. Dans l'exemple suivant, `QuelEstLeMax` exploite la fonction `Max` d'Excel, qui renvoie la valeur la plus forte dans une liste d'arguments. Si vous ne faites pas précéder `Max` du mot-clé `Application`, la fonction ne sera pas reconnue et une erreur sera générée.

```
Public Sub QuelEstLeMax()
  MsgBox "Le plus grand des chiffres reçus est " _
    & Application.Max(1,2,3,4), vbInformation + vbOKOnly, _
    "Utiliser les fonctions Excel dans VB"
End Sub
```

Créer des fonctions Excel personnalisées

Vous serez probablement amené à utiliser des fonctions qu'Excel ne prendra pas en charge. Le développement de procédures de type `Function` permet de remédier à ce problème.

Les fonctions personnalisées que vous créez dans Visual Basic Editor fonctionnent dans Excel comme n'importe quelle fonction intégrée. Elles apparaissent alors dans la liste des fonctions du tableur et peuvent être employées dans un classeur à l'aide de la commande Insérer une fonction de l'onglet Formules. Créez la fonction suivante dans Visual Basic Editor :

```
Function Agessa(SalaireBrut)
  Agessa = (SalaireBrut * 0.95) * 0.0085
End Function
```

Cette fonction calcule la cotisation perçue sur les droits d'auteur par l'AGESSA : une base de 95 % du salaire brut, au taux de 0,85 %.

Retournez dans Excel. Choisissez la commande Insérer une fonction du menu Formule et sélectionnez la catégorie Personnalisées. La fonction apparaît dans la liste Sélectionnez une fonction, précédée du nom du classeur dans lequel elle est stockée. Les arguments sont aussi visibles (voir figure 8-1).

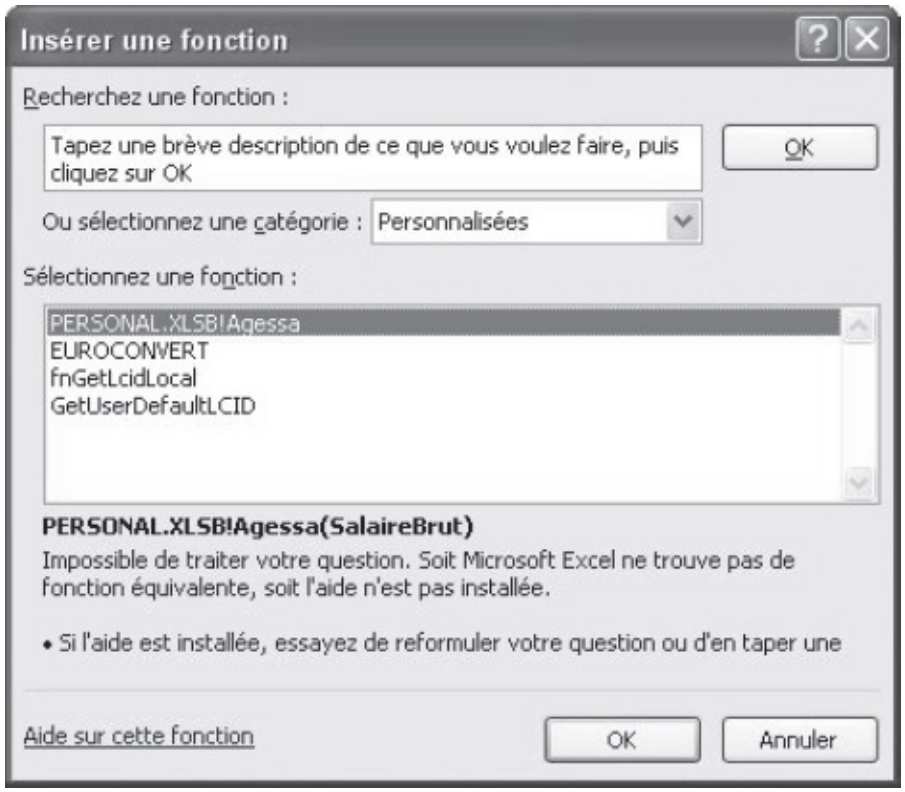

Insérer une fonction
Recherchez une fonction :
Tapez une brève description de ce que vous voulez faire, puis cliquez sur OK
OK
Ou sélectionnez une catégorie : Personnalisées
Sélectionnez une fonction :
PERSONAL.XLSB!Agessa
EUROCONVERT
fnGetLcidLocal
GetUserDefaultLCID
PERSONAL.XLSB!Agessa(SalaireBrut)
Impossible de traiter votre question. Soit Microsoft Excel ne trouve pas de fonction équivalente, soit l'aide n'est pas installée.
• Si l'aide est installée, essayez de reformuler votre question ou d'en taper une
Aide sur cette fonction
OK
Annuler

Figure 8-1 – *Les fonctions créées dans Visual Basic Editor peuvent être exploitées dans Excel, comme des fonctions intégrées.*

Les fonctions créées dans Visual Basic Editor sont gérées dans Excel comme n'importe quelle fonction intégrée du tableur. Si vous cliquez sur le bouton OK de la boîte de dialogue Insérer une fonction, vous serez invité à préciser les cellules correspondant aux arguments de la fonction (voir figure 8-2). La fonction apparaîtra ensuite dans la barre de formule.

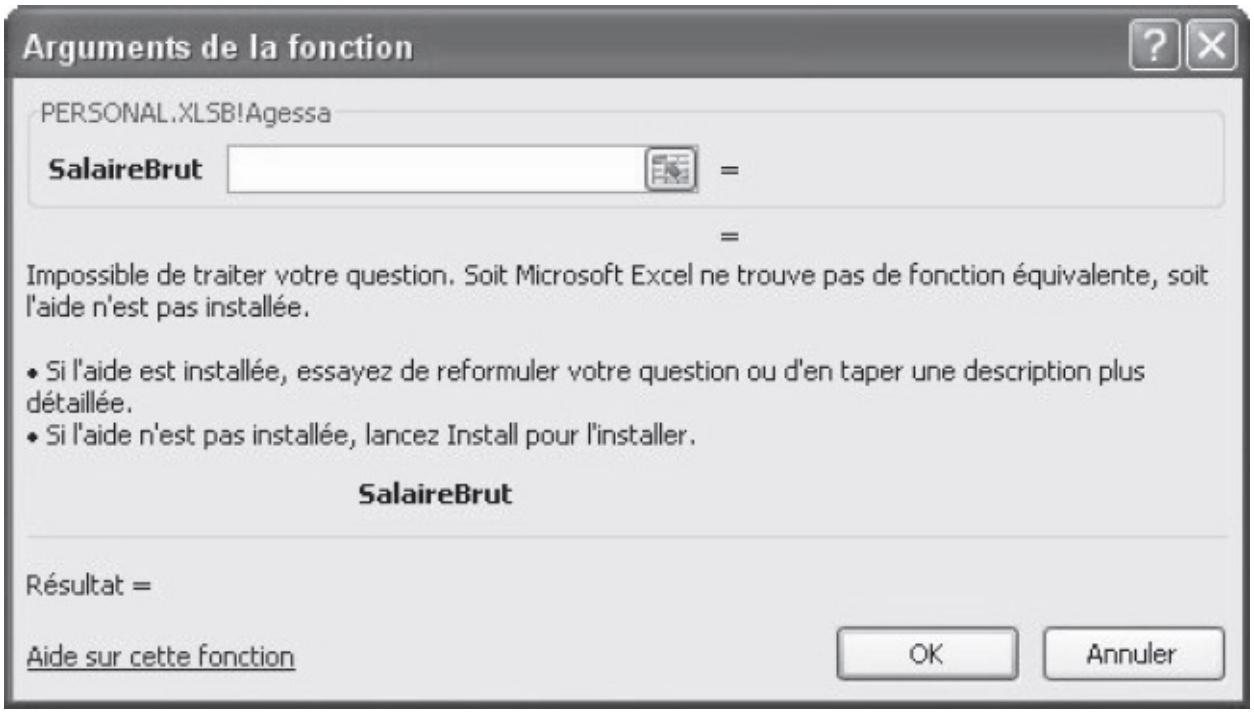

Arguments de la fonction
PERSONAL.XLSB!Agessa
SalaireBrut
=
=
Impossible de traiter votre question. Soit Microsoft Excel ne trouve pas de fonction équivalente, soit l'aide n'est pas installée.
• Si l'aide est installée, essayez de reformuler votre question ou d'en taper une description plus détaillée.
• Si l'aide n'est pas installée, lancez Install pour l'installer.
SalaireBrut
Résultat =
Aide sur cette fonction
OK
Annuler

Figure 8-2 – *Vous êtes invité à spécifier les cellules correspondant aux arguments de la fonction.*

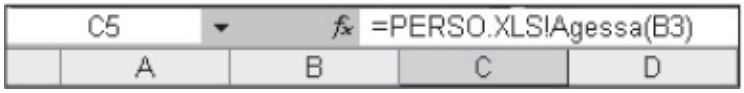


Figure 8-3 – *La fonction personnalisée s'affiche dans la barre de formule.*

Conseil

Affectez des noms représentatifs aux arguments des fonctions que vous créez, afin qu'ils soient compréhensibles pour les autres utilisateurs.

Intégrer une fonction via l'Explorateur d'objets

Au vu des centaines de fonctions proposées par Excel et accessibles dans VBA, il est difficile de mémoriser la syntaxe et les arguments de l'ensemble de ces fonctions. L'Explorateur d'objets est alors la solution. Il recense en effet les fonctions Excel et VBA – intégrées comme personnalisées – et en détaille l'usage et la syntaxe.

Insérer une fonction VBA dans votre code

Pour insérer une fonction VBA dans votre code à partir de l'Explorateur d'objets, procédez comme suit :

1. Lancez l'Explorateur d'objets.
2. Dans la liste déroulante Projet/Bibliothèque, choisissez VBA.
3. La zone Classes affiche la liste des objets, fonctions, procédures et constantes de VBA. Sélectionnez la catégorie (`Conversions`, `DateTime`, `Financial`, `Math`, `Strings`, etc.).
4. Dans la zone Membres de, sélectionnez la fonction.

   Les détails de la fonction apparaissent dans la zone inférieure de l'Explorateur d'objets. Si vous souhaitez davantage de précisions, cliquez sur le bouton Aide.

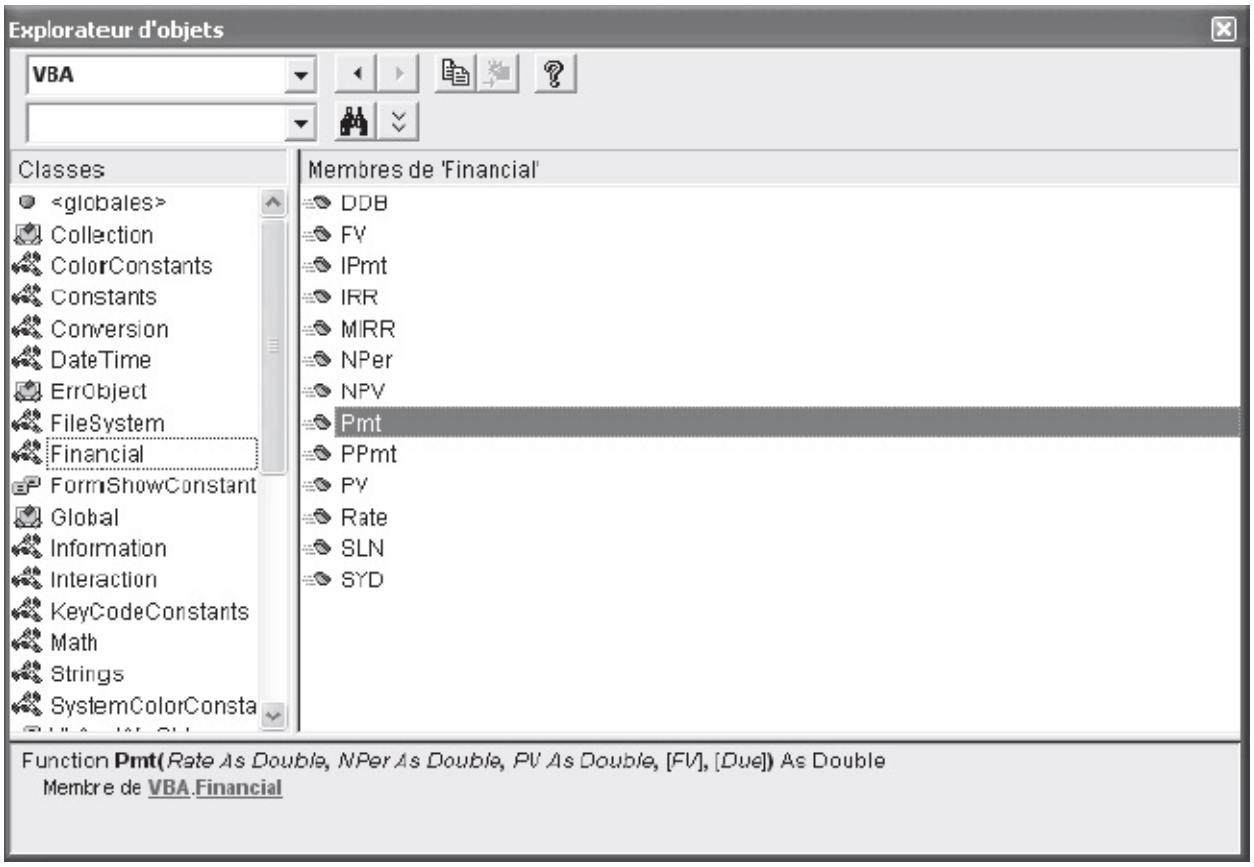


Figure 8-4 – *L'Explorateur d'objets donne accès à l'ensemble des informations disponibles pour une fonction.*

5. Cliquez ensuite sur le bouton Copier dans le Presse-papiers, puis fermez l'Explorateur d'objets.
6. Dans la fenêtre Code, placez le curseur à l'endroit souhaité, puis tapez le raccourci Ctrl+V afin de coller la fonction. Lorsque vous saisissez un espace, la liste des arguments attendus s'affiche.

Insérer une fonction Excel dans votre code

Pour insérer une fonction Excel dans votre code, procédez comme suit :

1. Lancez l'Explorateur d'objets.
2. Dans la liste Projet/Bibliothèque, choisissez Excel. Dans la zone Classes, choisissez `WorksheetFunction`.

   Les fonctions Excel disponibles dans VBA apparaissent dans la zone Membres de.

3. Sélectionnez la fonction voulue, puis copiez-la.
4. Placez ensuite le curseur à l'endroit où vous souhaitez faire apparaître la fonction dans la fenêtre Code.
5. Saisissez `Application.WorksheetFunction.`, puis collez la fonction.

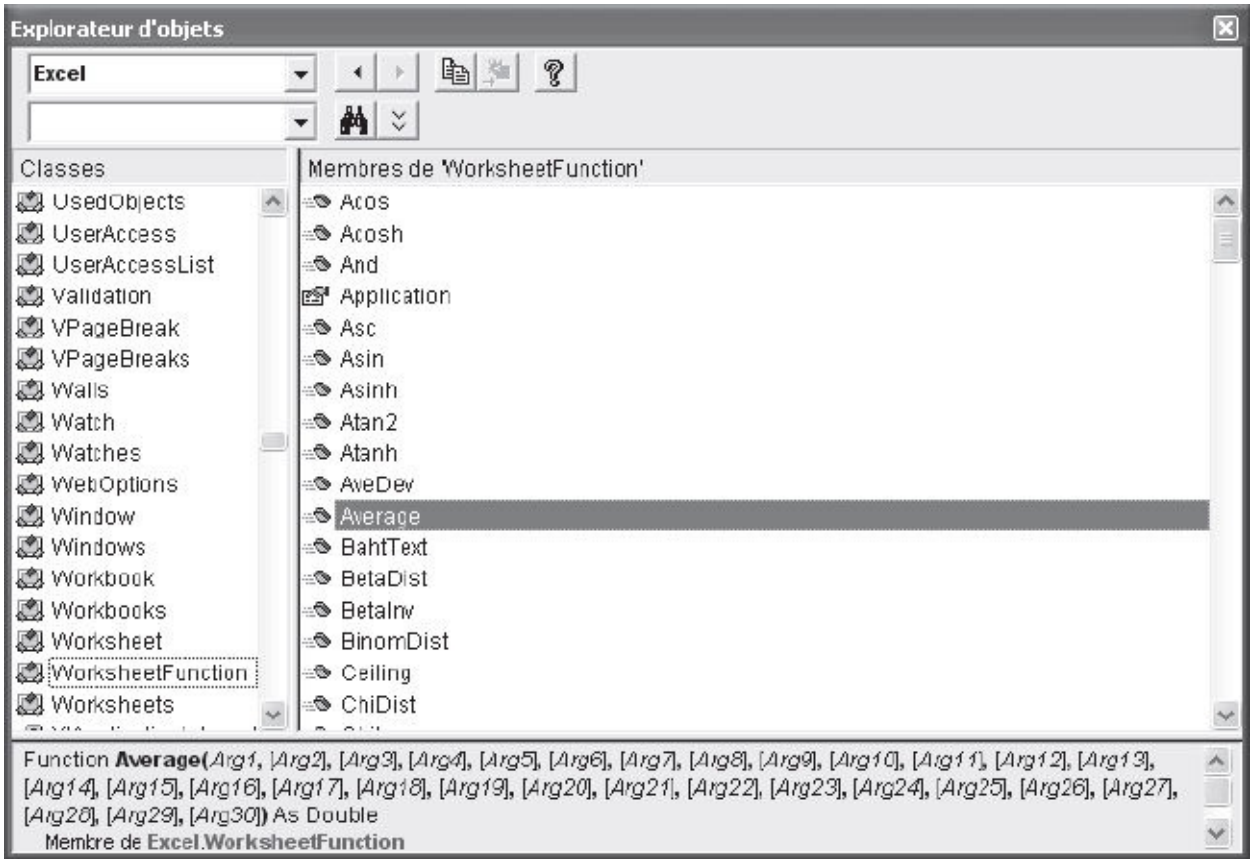


Figure 8-5 – *Les fonctions Excel sont accessibles via l'objet WorksheetFunction.*

Astuce

Sans passer par l'Explorateur d'objets, vous pouvez simplement saisir `Application.WorksheetFunction` dans votre code pour faire apparaître la liste des fonctions disponibles.

Notez que vous pouvez omettre la propriété `WorksheetFunction` et vous contenter de faire précéder la fonction de la propriété `Application`. Le code fonctionnera également, mais VBA n'affichera pas d'aide à l'écriture lors de la saisie dans la fenêtre Code.

Recommandations pour l'écriture de fonctions Excel

Les limites de la cellule

N'oubliez pas que vos fonctions serviront en général à insérer des données dans des cellules. Elles doivent donc présenter les mêmes limites que celles qui s'appliquent aux cellules. Veillez donc à ce que vos fonctions ne retournent jamais une chaîne supérieure à 255 caractères. Si c'était le cas, le résultat retourné et inséré dans la cellule serait tronqué au-delà de cette limite.

Précisez le type du résultat

Veillez à toujours préciser le type de données renvoyé par une fonction, afin que le résultat s'affiche correctement dans la cellule. Ceci est particulièrement vrai dans le cas d'une fonction renvoyant une date ; Excel n'appliquera en effet le format adéquat à la cellule que si la fonction a été déclarée comme retournant une valeur de type `Date`.

Des fonctions toujours à jour

Par défaut, Excel met à jour le résultat d'une fonction personnalisée dans une cellule. Vous pouvez cependant déclarer une fonction *volatile* de façon qu'Excel calcule ou recalcule le résultat d'une fonction chaque fois que le contenu d'une cellule de la feuille est modifié.

Pour marquer une fonction personnalisée comme volatile, insérez simplement l'instruction suivante immédiatement après la déclaration :

```
Application.Volatile
```

Considérez la fonction `CalculTauxBenefice` suivante. Elle retourne le taux de bénéfice réalisé à partir du chiffre d'affaires et des coûts :

```
Public Function TauxBenefice(CA As Currency, Couts As Currency) As Long
  Application.Volatile
  TauxBenefice = ((CA - Couts) / CA) * 100
End Function
```

Notre fonction reçoit les arguments de type `Currency` `CA` (pour chiffre d'affaires) et `Couts`. La première instruction de la fonction la déclare comme *volatile*. La fonction calcule ensuite le taux de bénéfice réalisé en pourcentage.

Principales fonctions VBA

Cette section présente les fonctions de calcul VBA les plus couramment utilisées dans le cadre de la programmation Excel. Elles sont classées selon les

catégories suivantes :

- mathématiques ;
- conversion de données ;
- conversion de types de données ;
- date et heure ;
- fonctions financières.

Pour obtenir une aide plus développée concernant leur utilisation, consultez l'aide en ligne de VBA.

Info

Les fonctions de traitement des chaînes de caractères constituant une question clé de la programmation, le chapitre suivant leur est entièrement consacré.

Info

Notez que les arguments des fonctions présentées dans les tableaux suivants sont pour la plupart des arguments nommés. Lorsque ce n'est pas le cas, le nom est traduit en français.

Dans le tableau 8-1, l'argument nommé *Number* représente une valeur numérique passée en argument.

Tableau 8-1. Fonctions mathématiques

Fonction	Description	Type de données renvoyé
`Abs(`*`Number`*`)`	Retourne la valeur absolue de *Number*.	Même type que *Number*
`Atn(`*`Number`*`)`	Retourne l'arctangente de *Number*, sous la forme d'un angle exprimé en radians. Pour convertir des degrés en radians, multipliez-les par pi/180. Inversement, pour convertir des radians en degrés, multipliez-les par 180/pi.	`Double`
`Cos(`*`Number`*`)`	Retourne le cosinus de l'angle *Number* (exprimé en radians).	`Double`
`Exp(`*`Number`*`)`	Retourne la valeur de la constante *e* élevée à la puissance *Number*. La constante *e* est la base des logarithmes népériens ; elle est à peu près égale à 2,718282.	`Double`
`Fix(`*`Number`*`)`	Renvoie la partie entière d'un nombre. Si *Number* est négatif, `Fix` renvoie le premier entier négatif supérieur ou égal à *Number*.	Même type que *Number*
	Comme `Fix`, `Int` renvoie la partie entière d'un	

`Int(Number)`	nombre, à la différence que, si *`Number`* est négatif, `Int` renvoie le premier entier négatif inférieur ou égal à *`Number`*.	Même type que *`Number`*
`Log(Number)`	Retourne le logarithme népérien de *`Number`*.	`Double`
`Rnd(Number)`	Retourne une valeur aléatoire. L'argument *`Number`* est facultatif et définit le comportement de la fonction `Rnd`. Notez que `Rnd` génère la même série de nombres aléatoires à chaque appel, car elle réutilise le nombre aléatoire précédent comme valeur initiale pour le calcul du nombre suivant. Utilisez l'instruction `Randomize` pour initialiser le générateur de nombres aléatoires à partir d'une valeur initiale tirée de l'horloge système.	`Single`
`Round(Number, NumDigitsAfterDecimal)`	Retourne la valeur arrondie de *`Number`*. L'argument facultatif *`NumDigitsAfterDecimal`* indique le nombre de chiffres à conserver après la virgule dans le nombre retourné. Si cet argument est omis, `Round` renvoie la valeur entière arrondie.	Même type que *`Number`*
`Sgn(Number)`	Retourne une valeur représentant le signe de *`Number`* : –1 si *`Number`* est inférieur à zéro ; 0 si *`Number`* est égal à zéro ; 1 si *`Number`* est supérieur à zéro.	`Integer`
`Sin(Number)`	Retourne le sinus de l'angle *`Number`* exprimé en radians.	`Double`
`Sqr(Number)`	Retourne la racine carrée de *`Number`*.	`Double`
`Tan(Number)`	Retourne la valeur de la tangente de l'angle *`Number`* exprimée en radians.	`Double`

Le tableau 8-2 présente les fonctions de conversion de données de VBA. Elles traitent la valeur reçue en argument et renvoient le résultat de ce traitement.

Les fonctions de conversion de types de données sont présentées dans le tableau 8-2.

Tableau 8-2. Fonctions de conversion des données

Fonction	Description	Type de données renvoyé
`Asc(String)`	Renvoie une valeur représentant le code du premier caractère de la chaîne de caractères *`String`*.	`Integer`
`Chr(CharCode)`	Renvoie le caractère dont le code est *`CharCode`*, de type `Long`.	`Integer`
`Format(Expression,`	Renvoie *`Expression`* sous forme de chaîne formatée selon le *`Format`* défini. Vous pouvez ainsi définir le format d'une date, ou	

Format)	modifier une chaîne pour afficher le symbole de la monnaie et placer des séparateurs entre les milliers.	String
Hex(*Number*)	Retourne la valeur hexadécimale de *Number* sous forme de chaîne. Si *Number* n'est pas un nombre entier, il est arrondi à l'entier le plus proche.	String
Oct(*Number*)	Retourne la valeur octale de *Number* sous forme de chaîne.	String
RGB(*Red, Green, Blue*)	Retourne un entier représentant la couleur. Les arguments *Red*, *Green* et *Blue* sont de type Integer et correspondent chacun à la valeur de la couleur associée (rouge, vert, bleu), comprise entre 0 et 255.	Long
Str(*Number*)	Renvoie *Number* sous forme de chaîne de caractères. *Number* peut être n'importe quelle valeur numérique que l'on souhaite traiter comme une chaîne.	String
Val(*String*)	Retourne *String* sous forme de valeur numérique de type approprié. Si *String* ne peut être converti en valeur numérique, une erreur est générée.	Type approprié à la valeur retournée

Reportez-vous au tableau 6-5 pour un aperçu des fonctions de conversion de données. Ces fonctions permettent de modifier le type d'une donnée. La conversion de données dans le type approprié est souvent nécessaire au bon déroulement d'un programme VBA. Par exemple, tenter d'exécuter une fonction mathématique sur une chaîne de caractères (même si celle-ci n'est composée que de chiffres) générera une erreur. Vous devrez alors faire appel à la fonction de conversion numérique appropriée afin que l'argument passé à la fonction soit reconnu comme une valeur et non plus comme une suite de caractères.

Le tableau 8-3 présente les fonctions de date et d'heure. L'argument nommé *date* désigne toute expression valide qui renvoie une valeur de type Date.

Tableau 8-3. Fonctions de date et d'heure

Fonction	Description	Type de données renvoyé
Date	Retourne la date en cours à partir de l'horloge système de votre ordinateur. Utilisez l'instruction Date pour redéfinir la date de l'horloge système.	Date
Time	Retourne l'heure en cours à partir de l'horloge système. Utilisez l'instruction Time pour redéfinir l'heure système.	Date
Now	Retourne la date et l'heure en cours.	Date
Year(*Date*)	Retourne l'année correspondant à *Date* sous la forme d'un	Integer

	nombre entier.	
`Month(Date)`	Retourne le mois correspondant à *`Date`* sous la forme d'un entier compris entre 1 (janvier) et 12 (décembre).	`Integer`
`Day(Date)`	Retourne le jour correspondant à *`Date`* sous la forme d'un nombre entier compris entre 1 et 31.	`Integer`
`Weekday(Date, FirstDayOfWeek)`	Retourne un entier compris entre 1 et 7, qui représente le jour de la semaine correspondant à *`Date`*. *`FirstDayOfWeek`* définit le jour considéré comme le premier de la semaine. Il peut s'agir d'une valeur comprise entre 0 et 7, ou de l'une des huit constantes `vbDayOfWeek` : `vbUseSystem`, `vbMonday`, `vbTuesday`, `vbWednesday`, `vbThursday`, `vbFriday`, `vbSaturday` et `vbSunday`. Si l'argument *`FirstDayOfWeek`* est omis, la valeur par défaut est `vbUseSystem` et dépend donc du système sur lequel est exécutée la macro. En France, le premier jour de la semaine par défaut est le lundi, tandis qu'aux États-Unis il s'agira du dimanche. Il est donc recommandé de préciser l'argument *`FirstDayOfWeek`* afin d'éviter des différences de comportements du programme d'un ordinateur à l'autre.	`Integer`
`WeekdayName (WeekDay, Abbreviate, FirstDayOfWeek)`	Retourne une chaîne qui représente le nom du jour de la semaine *`WeekDay`*. *`WeekDay`* peut être un nombre compris entre 0 et 7, ou l'une des constantes `vbDayOfWeek` présentées ci-avant. La valeur retournée est renvoyée dans la langue du système sur lequel est exécutée la fonction. L'argument nommé *`Abbreviate`* est facultatif et définit si le jour est renvoyé sous une forme abrégée (lun. pour lundi). Sa valeur par défaut est `False`. *`FirstDayOfWeek`* définit quel jour est considéré comme le premier de la semaine. Notez que la fonction `Weekday` peut être utilisée comme argument de `WeekdayName` : `… WeekDayName(WeekDay(Date))`	`String`
`Hour(Date)`	Retourne un entier compris entre 0 et 23, qui représente les heures de *`Date`*. Si *`Date`* ne fournit pas d'information d'heure, la fonction renvoie 0.	`Integer`
`Minute(Date)`	Retourne un entier compris entre 0 et 59, qui représente les minutes de *`Date`*. Si *`Date`* ne fournit pas d'information sur les minutes, la fonction renvoie 0.	`Integer`
`Second(Date)`	Retourne un entier compris entre 0 et 59, qui représente les secondes de *`Date`*. Si *`Date`* ne fournit pas d'information sur les secondes, la fonction renvoie 0.	`Integer`
`DateSerial(Year, Month, Day)`	Retourne une date précise (jour, mois et année), fonction des arguments *`Year`*, *`Month`* et *`Day`* reçus.	`Date`
`TimeSerial(Hour, Minute, Second)`	Retourne une heure précise (heures, minutes et secondes), en fonction des arguments reçus.	`Date`
`DateValue(Date)`	Renvoie la date reçue en argument. L'argument *`Date`* peut être une chaîne de caractères, un nombre, une constante ou n'importe quelle expression renvoyant une date.	`Date`

`TimeValue(Date)`	Renvoie l'heure reçue en argument. L'argument *`Date`* peut être une chaîne de caractères, un nombre, une constante ou n'importe quelle expression renvoyant une heure.	`Date`
`Timer`	Renvoie le nombre de secondes écoulées depuis minuit (d'après l'horloge système de votre ordinateur).	`Single`

Le tableau 8-4 présente les fonctions financières de VBA.

Tableau 8-4. Fonctions financières

Fonction	Description	Type de données renvoyé
Fonctions de calcul d'amortissement		
`DDB(Cost, Salvage, Life, Period,)`	Retourne la valeur de l'amortissement d'un bien au cours d'une période définie, en utilisant la méthode d'amortissement dégressif à taux double ou toute autre méthode précisée. Les arguments nommés reçus sont : – *`Cost`* : coût initial du bien. – *`Salvage`* : valeur du bien à la fin de son cycle de vie. – *`Life`* : durée du cycle de vie du bien. – *`Period`* : période sur laquelle l'amortissement du bien est calculé. – *`Factor`* : facultatif. Taux utilisé pour le calcul de l'amortissement. Si *`Factor`* est omis, la valeur 2 est employée (méthode d'amortissement dégressif à taux double).	`Double`
`SLN(Cost, Salvage, Life)`	Retourne l'amortissement linéaire d'un bien sur une période définie. Les arguments nommés sont les mêmes que pour la fonction `DDB()`.	`Double`
`SYD(Cost, Salvage, Life, Period)`	Retourne la valeur de l'amortissement global d'un bien sur une période donnée. Les arguments nommés sont les mêmes que pour la fonction `DDB()`.	`Double`
Fonctions de calcul de valeur future		
`FV(Rate, NPer, Pmt, PV, Type)`	Retourne le futur montant d'une annuité basée sur des versements constants et périodiques et sur un taux d'intérêt fixe. Les arguments nommés reçus sont : – *`Rate`* : taux d'intérêt par échéance. Pour un prêt dont le taux annuel serait de 5 %, avec des échéances mensuelles, le taux par échéance serait de 0,05/12, soit 0,0042 ; – *`NPer`* : nombre d'échéances de l'emprunt ou du placement ; – *`Pmt`* : montant du paiement à effectuer à chaque échéance ; – *`PV`* : facultatif. Valeur actuelle (ou somme globale) d'une série de paiements devant être effectués dans le futur. La valeur par défaut est 0 ; – *`Type`* : facultatif. Date d'échéance des paiements. Indiquez 0 si les paiements sont dus à terme échu, ou 1 si les paiements sont dus à terme à échoir. Si l'argument est omis, la valeur par défaut est 0.	`Double`

Fonctions de calcul de taux d'intérêt		
	Retourne le taux d'intérêt par échéance pour une annuité.	
`Rate(NPer, Pmt, PV, FV, Type, Guess)`	Les arguments nommés *NPer*, *Pmt*, *PV* et *Type* sont les mêmes que pour la fonction `FV()`. – *FV* : facultatif. Valeur future ou solde que vous souhaitez obtenir après avoir effectué le dernier paiement. Si vous souhaitez épargner 10 000 ‡, la valeur future est de 10 000. Si l'argument est omis, la valeur par défaut est 0 (qui est la valeur future d'un emprunt). – *Guess* : facultatif. Valeur qui devrait être renvoyée par la fonction `Rate`. S'il est omis, *Guess* prend la valeur 0,1 (10 pour cent).	`Double`
Fonctions de calcul de taux de rendement interne		
`IRR(Values(), Guess)`	Retourne le taux de rendement interne d'une série de mouvements périodiques de trésorerie (paiements et encaissements). Les arguments nommés sont les suivants : – *Values()* : tableau de données indiquant les valeurs des mouvements de trésorerie. Le tableau doit contenir au moins une valeur négative (un paiement) et une valeur positive (un encaissement) ; – *Guess* : facultatif. Valeur qui devrait être renvoyée par la fonction `IRR()`. S'il est omis, *Guess* prend pour valeur 0,1 (10 pour cent).	`Double`
`MIRR(Values(), Finance_Rate, Reinvest_Rate)`	Retourne le taux de rendement interne modifié d'une série de mouvements périodiques de trésorerie (paiements et encaissements) : – *Values()* : idem que pour la fonction `IRR()` ; – *Finance_Rate* : taux d'intérêt payé pour couvrir le coût du financement ; – *Reinvest_Rate* : taux d'intérêt perçu sur les gains tirés des sommes réinvesties.	`Double`
Fonctions de calcul de nombre d'échéances		
`NPer(Rate, Pmt, PV, FV, Type)`	Retourne le nombre d'échéances d'une annuité basée sur des versements constants et périodiques et sur un taux d'intérêt fixe. Les arguments nommés sont les mêmes que pour les fonctions `FV()` et `Rate()`.	`Double`
Fonctions de calcul de montant de versements		
`IPmt(Rate, per, NPer, PV, FV, Type)`	Renvoie une valeur indiquant le montant, sur une période donnée, d'une annuité basée sur des versements constants et périodiques et sur un taux d'intérêt fixe. Les arguments nommés sont les mêmes que pour les fonctions `FV()` et `Rate()`.	`Double`
`Pmt(Rate, NPer, PV, FV, Type)`	Retourne le montant d'une annuité basée sur des versements constants et périodiques et sur un taux d'intérêt fixe. Les arguments nommés sont les mêmes que pour les fonctions `FV()` et `Rate()`.	`Double`
	Retourne la valeur du remboursement du capital, pour une	

PPmt(*Rate, per, NPer, PV, FV, Type*)	échéance donnée, d'une annuité basée sur des versements constants et périodiques, et sur un taux d'intérêt fixe. Les arguments nommés sont les mêmes que pour les fonctions FV() et Rate().	Double
Fonctions de calcul de montant de valeur actuelle		
NPV(*Rate, Values()*)	Retourne la valeur nette actuelle d'un investissement, calculée en fonction d'une série de mouvements périodiques de trésorerie (paiements et encaissements) et d'un taux d'escompte. Les arguments nommés sont les suivants : – *Rate* : taux d'escompte sur la période, exprimé sous la forme d'un nombre décimal ; – Values() : tableau de données indiquant les valeurs des mouvements de trésorerie. Le tableau doit contenir au moins une valeur négative (un paiement) et une valeur positive (un encaissement).	Double
PV(*Rate, NPer, Pmt, FV, Type*)	Retourne le montant actuel d'une annuité basée sur des échéances futures constantes et périodiques, et sur un taux d'intérêt fixe. Les arguments nommés sont les mêmes que pour les fonctions FV() et Rate().	Double

9

Manipuler des chaînes de caractères

La manipulation de chaînes de caractères est un aspect clé de la programmation, quel que soit le langage, notamment pour informer et interagir avec l'utilisateur. Dans ce cadre, les valeurs numériques doivent souvent être traitées et formatées comme des chaînes de caractères afin d'en extraire les informations pertinentes. Par ailleurs, les informations retournées par la fonction `InputBox` ou *via* un contrôle `TextBox` sont des chaînes de caractères et devront être traitées comme telles.

Modifier des chaînes de caractères

De nombreuses possibilités de manipulations s'offrent à vous. Vous pouvez bien sûr ajouter des éléments à une chaîne de caractères en la concaténant avec d'autres, mais aussi en extraire une partie de façon à n'en conserver que l'information pertinente.

Concaténer des chaînes

Concaténer des chaînes consiste à les « accoler », c'est-à-dire à en assembler plusieurs pour en créer une seule. Il est également possible de concaténer les éléments d'un tableau, de façon à obtenir une chaîne qui en reprenne toutes les valeurs.

Concaténer des chaînes simples

La concaténation des chaînes est réalisée à l'aide des opérateurs `&` et `+`. Il est cependant conseillé de privilégier l'opérateur `&`, qui ne fonctionne qu'avec les chaînes, et de conserver le `+` pour les additions. Votre code sera ainsi sans ambiguïté.

Toute expression qui renvoie une chaîne de caractères est utilisable : variable de type `String`, fonction renvoyant une chaîne, chaîne de caractères entourée de

guillemets.

> **Attention**
> Si vous souhaitez inclure une valeur numérique dans une chaîne, utilisez la fonction `Str()` pour la convertir.

Considérez l'exemple suivant :

```
1: Sub ConcatenerDesChaînes()
2:   Dim MonNom As String
3:   Dim MonBenefice As Long
4:   Dim MonMessage As String
5:   MonNom = ActiveWorkbook.ActiveSheet.Cells(1,2).Value
6:   MonBenefice = ActiveWorkbook.ActiveSheet.Cells(1,3).Value
7:   MonMessage = "La prime de " & MonNom & " est de " & Str(Int(MonBenefice/20))
      ➥ & " euros."
8:   MsgBox MonMessage
9: End Sub
```

Lignes 2 à 4, les variables sont déclarées. Notez que `MonBenefice` est de type `Long` et ne peut donc être concaténée telle quelle. Lignes 5 et 6, `MonNom` et `MonBenefice` reçoivent respectivement pour valeur le contenu des cellules B1 et C1.

Ligne 7 a lieu la concaténation des différentes chaînes de façon à générer le message affiché par la fonction `MsgBox` à la ligne suivante.

Les éléments concaténés sont les suivants :

- `"La prime de "` est une simple chaîne de caractères, entourée de guillemets.
- `MonNom` est une variable de type `String` et peut donc être concaténée sans qu'il soit nécessaire d'effectuer une conversion de type de données.
- `" est de "` est une simple chaîne de caractères. Notez que nous avons fait précéder et suivre le texte d'espaces, pour que la chaîne finale soit lisible.
- `Str(Int(MonBenefice/20))` est une expression renvoyant une chaîne de caractères. Tout d'abord, la valeur numérique `MonBenefice` est divisée par 20, de façon à obtenir la valeur de la prime (5 % du bénéfice). La fonction `Int` renvoie la valeur entière de la somme ainsi obtenue. Enfin, la fonction `Str` convertit le résultat numérique ainsi obtenu en une chaîne de caractères.
- `" euros"` est une simple chaîne de caractères.

On obtient ainsi le message affiché dans la boîte de dialogue de la figure 9-1.

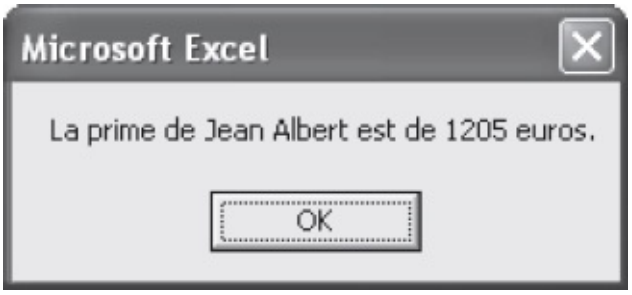


Figure 9-1 – *La concaténation de chaînes permet de construire des messages pour l'utilisateur.*

Concaténer les valeurs d'une variable de matrice

Si vous souhaitez créer une seule chaîne de caractères à partir des différentes valeurs d'un tableau, utilisez la fonction `Join` selon la syntaxe suivante :

```
Join(SourceArray, Delimiter)
```

`Delimiter` (facultatif) définit la chaîne de caractères qui sera utilisée comme séparateur entre les différentes données extraites de la *`SourceArray`*. S'il est omis, les éléments du tableau sont concaténés sans séparateur.

Dans l'exemple suivant, les jours de la semaine sont stockés dans le tableau `JoursSemaine`. La boîte de dialogue représentée à la figure 9-2 est ensuite affichée. Notez qu'on utilise une virgule suivie d'un espace comme séparateur et que l'on ajoute un point à la fin de la chaîne.

```
Sub ConcatenerUnTableau()
  Dim JoursSemaine(1 To 7)
  Dim n as Byte
  For n = 1 To 7
    JoursSemaine(n) = WeekdayName(Weekday:=n, abbreviate:=False, _
      FirstDayOfWeek:=vbMonday)
  Next n
  MsgBox "Les jours de la semaine sont : " & Join(JoursSemaine, ", ") & "."
End Sub
```

Figure 9-2 – *Utilisez la fonction Join pour concaténer les espaces de stockage d'un tableau.*

Insérer des caractères non accessibles au clavier

Pour insérer, dans des chaînes, des caractères qui ne peuvent être saisis dans le code VBA à l'aide du clavier – tels qu'un saut de paragraphe ou un retour à la ligne –, on fait appel à la fonction `Chr()` qui renvoie le caractère dont le code ASCII est précisé entre parenthèses, selon la syntaxe suivante :

```
Chr(CharCode)
```

Dans l'exemple suivant, on insère un retour chariot, `Chr(13)`, et une puce, `Chr(149)`, devant chaque jour de la semaine, afin de générer le message ensuite affiché dans une boîte de dialogue (voir figure 9-3).

```
Sub UtiliserChr()
   Dim n As Byte
   Dim Message As String
   Message = "Les jours de la semaine sont :"
   For n = 1 To 7
     Message = Message & Chr(13) & Chr(149) & "  " & WeekdayName(Weekday:=n,
       ➥ abbreviate:=False, FirstDayOfWeek:=vbMonday)
   Next n
   MsgBox Message
End Sub
```

Figure 9-3 – *Utilisez la fonction Chr() pour insérer des caractères non accessibles au clavier.*

Afin d'améliorer la lisibilité de votre code, préférez les constantes Visual Basic à la fonction `Chr()` quand cela est possible. Le tableau 9-1 présente les plus communément employées pour créer des chaînes de caractères.

Tableau 9-1. Constantes Visual Basic renvoyant un caractère

Constante	Valeur	Equivalent avec *Chr()*
vbCr	Saut de paragraphe	Chr(13)
vbLf	Saut de ligne	Chr(10)
vbCrLf	Retour chariot + saut de ligne	Chr(13) & Chr(10)
vbNewLine	Saut de ligne spécifique à la plate-forme : équivaut à vbCrLf pour Windows et à vbCr sur un Macintosh	Chr(13) & Chr(10) pour Windows Chr(13) pour Macintosh
vbTab	Tabulation	Chr(9)

Répéter une série de caractères

Utilisez la fonction String() pour renvoyer une chaîne composée d'un caractère répété un certain nombre de fois, selon la syntaxe suivante :

```
String(Number, Character)
```

où les arguments nommés *Number* et *Character* représentent respectivement le nombre de répétitions et le caractère à répéter. L'expression suivante insère dans une chaîne la valeur de la cellule A1 à laquelle sont ajoutés six zéros. La valeur, exprimée en millions dans la cellule, est ainsi récupérée sous forme d'unités :

```
MsgBox Str(ActiveWorkbook.ActiveSheet.Range("A1").Value) & String(6,"0")
```

Astuce

Vous pouvez utiliser la fonction Chr() pour l'argument Character. Ainsi String(10,Chr(149)) renvoie une chaîne composée de dix puces.

De façon similaire, la fonction Space renvoie une chaîne composée d'un nombre défini d'espaces, selon la syntaxe suivante :

```
Space(Number)
```

Supprimer les espaces superflus d'une chaîne

Traiter les espaces d'une chaîne est une opération souvent indispensable au bon fonctionnement d'un programme. Par exemple, si vous recevez des informations de l'utilisateur grâce à la fonction InputBox, il sera prudent de supprimer les éventuels espaces superflus à l'une ou l'autre des extrémités de la chaîne retournée avant de traiter ces données.

Visual Basic propose trois fonctions pour ce faire :

- **`LTrim(`*`String`*`)`** retourne la chaîne *`String`* après en avoir supprimé les espaces situés à gauche.
- **`RTrim(`*`String`*`)`** retourne la chaîne *`String`* après en avoir supprimé les espaces situés à droite.
- **`Trim(`*`String`*`)`** retourne la chaîne *`String`* après en avoir supprimé les espaces situés à droite et à gauche.

L'instruction suivante stocke les données saisies par l'utilisateur dans `MaVar` après avoir pris soin de supprimer les éventuels espaces saisis par erreur :

```
MaVar = Trim(InputBox("Veuillez saisir votre nom : "))
```

Extraire une partie d'une chaîne

Les fonctions `Left()`, `Right()` et `Mid()`, combinées aux fonctions de recherche (présentées plus loin dans ce chapitre), servent à extraire une partie précise d'une chaîne de caractères. En argument, elles utilisent souvent `Len()` – qui renvoie la longueur d'une chaîne, c'est-à-dire le nombre de caractères qui la composent.

- **`Len(`*`String`*`)`** retourne une valeur de type `Double` qui représente le nombre de caractères présents dans la chaîne *`String`*.
- **`Left(`*`String`*`, `*`Length`*`)`** retourne les *`Length`* premiers caractères de *`String`*. Si *`Length`* est supérieur ou égal à la longueur de *`String`*, la totalité de la chaîne est retournée.
- **`Right(`*`String`*`, `*`Length`*`)`** retourne les *`Length`* derniers caractères de *`String`*. Si *`Length`* est supérieur ou égal à la longueur de *`String`*, la totalité de la chaîne est retournée.
- **`Mid(`*`String`*`, `*`Start`*`, `*`Length`*`)`** retourne *`Length`* caractères de *`String`* à partir de celui (inclus) de la position *`Start`*. Si l'argument *`Length`* (facultatif) est omis ou s'il est trop grand, la fonction renvoie tous les caractères de *`Start`* jusqu'à la fin de *`String`*.

Considérez les trois instructions suivantes :

```
1: MonTexte = Left(MonTexte, Len(MonTexte) - 1)
2: MonTexte = Right(MonTexte, Len(MonTexte) - 1)
3: Trim(Mid(ActiveWorkbook.ActiveSheet.Range("A1").Value, InStr(ActiveWorkbook.
   ➥ ActiveSheet.Range("A1").Value, Chr(34)) + 1))
```

La première instruction modifie la variable `MonTexte` en supprimant son dernier caractère. Ligne 2, on supprime le premier caractère de `MonTexte`.

Enfin, l'instruction de la ligne 3 retourne le contenu de la cellule A1 à partir du

premier guillemet (non inclus) présent dans la chaîne. On utilise pour cela la fonction `InStr()` présentée plus loin dans ce chapitre, qui retourne la position du caractère recherché (ici `Chr(34)`, soit un guillemet). On ajoute 1 à cette valeur pour commencer la chaîne à partir du caractère qui suit le guillemet. Notez que l'on emploie la fonction `Trim()` afin de supprimer les éventuels espaces présents aux extrémités de la chaîne retournée par la fonction `Mid()`.

Effectuer des remplacements au sein d'une chaîne

Utilisez la fonction `Replace()` pour effectuer des remplacements au sein d'une chaîne de caractères selon la syntaxe suivante :

```
Replace(Expression, Find, Replace, Start, Count, Compare)
```

`Expression` est une expression valide qui renvoie la chaîne de caractères à traiter. *`Find`* est la chaîne recherchée et à remplacer par *`Replace`*. Les trois arguments suivants sont facultatifs :

- ***`Start`*** précise la position à partir de laquelle doivent commencent les remplacements dans *`Expression`*. Si cet argument est omis, l'ensemble de la chaîne est traité.
- ***`Count`*** indique le nombre de remplacements à effectuer. S'il est omis, sa valeur par défaut –1 est utilisée et toutes les occurrences de *`Find`* sont remplacées.
- Enfin, *`Compare`* précise le type de comparaison effectuée. Il peut s'agir de l'une des valeurs suivantes :
  - **`vbUseCompareOption`** ou –1. C'est la valeur de l'option `Option Compare` précisée dans l'en-tête du module dans lequel se trouve l'instruction qui est utilisée pour définir le type de comparaison.
  - **`vbBinaryCompare`** ou 0. Effectue une comparaison binaire. La recherche respecte la casse de *`Find`*.
  - **`vbTextCompare`** ou 1. Effectue une comparaison de texte. La recherche ne prend pas en considération la casse des occurrences trouvées.
  - **`vbDatabaseCompare`** ou 2. Effectue une comparaison s'appuyant sur des informations contenues dans une base de données Microsoft Access.

Dans l'exemple suivant, lignes 7 à 9, les données de la feuille active sont stockées dans le tableau `MesValeurs`, tout en remplaçant au passage les « ; » par des « , ».

```
1: Sub UtiliserReplace()
```

```
 2:   Dim MaCellule As Range
 3:   Dim n As Long
 4:   Dim MesValeurs()
 5:   ReDim MesValeurs(1 To ActiveWorkbook.ActiveSheet.UsedRange.Cells.Count)
 6:   n = 0


 7:   For Each MaCellule In ActiveWorkbook.ActiveSheet.UsedRange.Cells
 8:     n = n + 1
 9:     MesValeurs(n) = Replace(MaCellule.Value, ";", ",")
10:   Next MaCellule
11: End Sub
```

Astuce

Affectez une chaîne nulle à l'argument `Replace` pour supprimer toutes les occurrences de `Find` dans la chaîne traitée.

Modifier la casse des chaînes de caractères

Les fonctions suivantes servent à formater une chaîne :

- **`StrConv(String, Conversion)`** convertit les majuscules et minuscules de la chaîne *`String`* selon la valeur de *`Conversion`* :
  - `vbUpperCase` ou 1 convertit la chaîne en majuscules.
  - `vbLowerCase` ou 2 convertit la chaîne en minuscules.
  - `vbProperCase` ou 3 convertit les lettres de la chaîne en minuscules et applique une majuscule à la première lettre de chaque mot.
- **`UCase(String)`** retourne la chaîne *`String`* en convertissant toutes les lettres minuscules en majuscules. Équivaut à utiliser la fonction `StrConv()` en affectant la valeur 1 ou `vbUpperCase` à l'argument *`Conversion`*.
- **`LCase(String)`** retourne la chaîne *`String`* en convertissant toutes les lettres majuscules en minuscules. Équivaut à utiliser la fonction `StrConv()` en affectant la valeur 2 ou la constante `vbLowerCase` à l'argument *`Conversion`*.

Comparer des chaînes de caractères

Comme vous le verrez en développant vos programmes VBA, la comparaison de chaînes de caractères est souvent utilisée dans des structures de contrôle pour répéter une opération jusqu'à ce qu'une condition soit remplie.

La méthode la plus simple pour comparer des chaînes consiste à employer les opérateurs arithmétiques =, < et >. Vous pouvez également appeler la fonction `StrComp()`, en respectant la syntaxe suivante :

```
StrComp(String1, String2, Compare)
```

L'argument facultatif *Compare* définit le type de comparaison à effectuer (voir les explications fournies pour la commande `Replace`) :

- **vbUseCompareOption** ou –1.
- **vbBinaryCompare** ou 0.
- **vbTextCompare** ou 1.
- **vbDatabaseCompare** ou 2.

La fonction `StrComp()` renvoie les valeurs suivantes :

- –1 : *String1* est inférieure à *String2*.
- 0 : *String1* est égale à *String2*.
- 1 : *String1* est supérieure à *String2*.
- Null : *String1* ou *String2* est de type `Null`.

La fonction `StrComp()` compare les codes ANSI des caractères qui composent les chaînes, par ordre de position. Lorsque deux caractères comparés ont des codes différents, le processus s'arrête et la fonction renvoie la valeur correspondante. Sinon la fonction passe aux caractères suivants des deux chaînes, etc.

Le tableau suivant présente quelques exemples de comparaisons de chaînes en mode de comparaison `vbTextCompare` :

String1	*String2*	Résultat de la comparaison
a	A	Les deux chaînes sont identiques.
abc	AbC	Les deux chaînes sont identiques.
ab	ba	*String1* est inférieure à *String2*.
abc	a	*String1* est supérieure à *String2*.

La procédure suivante fait appel à la fonction `InputBox()` pour afficher à deux reprises une boîte de saisie dans laquelle l'utilisateur est invité à entrer une chaîne de caractères. Les deux chaînes sont ensuite comparées et le résultat s'affiche dans une boîte de dialogue grâce à l'instruction `MsgBox`.

```
1: Sub ComparerDesChaînes()
2:   Dim MaChaîne1 As String
3:   Dim MaChaîne2 As String
4:   MaChaîne1 = InputBox("Veuillez entrer la valeur de la chaîne 1 :",
     ➥ "Comparaison de chaînes")
5:   Do Until MaChaîne1<>""
6:     MaChaîne1 = InputBox("Vous devez préciser une chaîne à comparer !" & _
7:       vbCr & "Entrez une valeur pour la chaîne 1 :", "Comparaison de chaînes")
8:   Loop
```

```
 9:   MaChaîne2 = InputBox("Veuillez entrer la valeur de la chaîne 2 :",
      ➥ "Comparaison de chaînes")
10:   Do Until MaChaîne2<>""
11:     MaChaîne2 = InputBox("Vous devez préciser une chaîne à comparer !" & _
12:       vbCr & "Entrez une valeur pour la chaîne 2 :", "Comparaison de chaînes")
13:   Loop
14:   Select Case StrComp(MaChaîne1, MaChaîne2, vbTextCompare)
15:     Case -1
16:       MsgBox "La chaîne 1 est inférieure à la chaîne 2."
17:     Case 0
18:       MsgBox "La chaîne 1 et la chaîne 2 sont identiques."
19:     Case 1
20:       MsgBox "La chaîne 1 est supérieure à la chaîne 2."
21:   End Select
22: End Sub
```

Lignes 5 à 8 pour la première chaîne et 10 à 13 pour la seconde, une structure `Do Until…Loop` affiche de nouveau une boîte de saisie si l'utilisateur a validé avec une zone de saisie vide. On utilise ici l'opérateur `<>` pour comparer deux chaînes (la variable et une chaîne vide) afin de définir si la boucle doit être exécutée.

Lignes 14 à 21, une structure `Select Case…End Select` définit le message à afficher selon la valeur renvoyée par l'instruction `StrComp()`.

Rechercher dans les chaînes de caractères

La recherche dans des chaînes de caractères est l'une des opérations les plus courantes. VBA permet de rechercher une chaîne de caractères au sein d'une autre chaîne, mais aussi dans des variables de matrice.

Rechercher une chaîne dans une chaîne

Pour rechercher une chaîne au sein d'une autre, faites appel à la fonction `InStr()`. Elle renvoie la position dans une chaîne *`string1`* de la première occurrence de la chaîne *`string2`* qui y est recherchée, selon la syntaxe suivante :

```
InStr(start, string1, string2, compare)
```

L'argument *`start`* est facultatif et indique la position du caractère à partir duquel commence la recherche (le premier par défaut). L'argument facultatif *`compare`* définit le type de comparaison à effectuer (voir les précisions fournies pour la commande `Replace`) :

- **`vbUseCompareOption`** ou –1.
- **`vbBinaryCompare`** ou 0.
- **`vbTextCompare`** ou 1.

- **`vbDatabaseCompare`** ou 2.

Dans l'exemple suivant, la procédure `MaMacro` appelle la fonction `VerifierEnregistrement` pour s'assurer que le document actif est enregistré avant de s'exécuter. Si ce n'est pas le cas, la fonction renvoie `False` et l'instruction `End` met fin à l'exécution du programme après avoir affiché un message à l'attention de l'utilisateur.

```
 1: Sub MaMacro()
 2:   If VerifierEnregistrement(ActiveWorkbook)=False Then
 3:     MsgBox "Cette commande ne peut être exécutée que sur un fichier
        ➥ enregistré.", _
 4:       vbOKOnly + vbCritical, "Exécution impossible"
 5:     End
 6:   End If
 7:   'suite des instructions de la macro
 8: End Sub


 9: Function VerifierEnregistrement(MonClasseur As Workbook) As Boolean
10:   If InStr(MonClasseur.FullName, Application.PathSeparator)=0 Then
11:     VerifierEnregistrement = False
12:   Else
13:     VerifierEnregistrement = True
14:   End If
15: End Function
```

La fonction `VerifierEnregistrement` est appelée ligne 2 et reçoit en argument le classeur actif. Ligne 10, on recherche le séparateur propre au système sur lequel s'exécute le programme (`Application.PathSeparator`) au sein du nom complet du document, chemin inclus (propriété `FullName`). Si le fichier n'a pas été enregistré, la chaîne retournée par `FullName` est la même que celle qui serait retournée par `Name` et ne contient pas de séparateur ; la fonction `InStr()` renvoie donc 0 et notre fonction reçoit la valeur `False`. Dans le cas contraire, c'est la valeur `True` qui lui est affectée.

Dans l'exemple suivant, la fonction `ScinderChaîne` est appelée par `MaMacro` et reçoit en arguments deux chaînes de caractères : celle qu'il faut scinder en plusieurs parties et le séparateur à utiliser. Elle renvoie une variable de matrice contenant les différents morceaux extraits. Ceux-ci sont ensuite traités en vue d'être affichés.

```
 1: Sub MaMacro()
 2:   Dim n As Long
 3:   Dim MonResultat()
 4:   Dim Exemple As String
 5:   Dim Message As String
 6:   Exemple = "Transports:Maritimes:Bateaux:A voile"
 7:   MonResultat = ScinderChaîne(Exemple,":")
 8:   For n = 1 To UBound(MonResultat)
 9:     Message = Message & vbCr & Chr(149) & _
         "  " & MonResultat(n)
10:   Next n
11:   MsgBox "La chaîne a été scindée en " & UBound(MonResultat) & _
```

```
       " parties : " & Message
12: End Sub


13: Function ScinderChaîne (machaîne As String, separateur As String)
14:   Dim pos As Long
15:   Dim mavar
16:   ReDim mavar(1)
17:   pos = InStr(machaîne, separateur)
18:   Do While pos<>0
19:     mavar(UBound(mavar)) =Left(machaîne,pos-1)
20:     machaîne = Mid(machaîne, pos+Len(separateur))
21:     pos = InStr(machaîne, separateur)
22:     ReDim Preserve mavar(UBound(mavar)+1)
23:   Loop
24:   mavar(UBound(mavar)) = machaîne
25:   ScinderChaîne = mavar
26: End Function
```

Ligne 7, la fonction `ScinderChaîne` est appelée et reçoit en arguments la variable `Exemple` définie à la ligne précédente, ainsi que le séparateur à utiliser, en l'occurrence deux-points.

La fonction `ScinderChaîne` prend alors la main. La variable de matrice `mavar` est déclarée puis initialisée à la ligne suivante. Ligne 17, `pos` reçoit le résultat de la recherche du séparateur au sein de la chaîne.

Lignes 18 à 23, une structure de contrôle `Do While…Loop` répète la recherche tant qu'elle aboutit, c'est-à-dire tant que la fonction `InStr()` renvoie une valeur différente de 0. À chaque nouvelle recherche, on affecte au dernier espace de stockage de `mavar` la chaîne précédant la position du séparateur trouvé (`Left(machaîne, pos-1)`). `machaîne` est ensuite redéfinie de façon à en supprimer la partie extraite, ainsi que le séparateur recherché (ligne 20), grâce aux fonctions `Mid()` et `Len()`. La comparaison est alors effectuée sur la nouvelle chaîne (ligne 21) et `mavar` reçoit un espace de stockage supplémentaire. Notez l'utilisation du mot-clé `Preserve` pour conserver les valeurs déjà stockées dans `mavar`.

Enfin, ligne 24, le dernier espace de stockage de `mavar` reçoit la valeur de `machaîne` quand la recherche n'a pas abouti. La procédure appelante reprend alors la main et le message représenté à la figure 9-4 s'affiche.

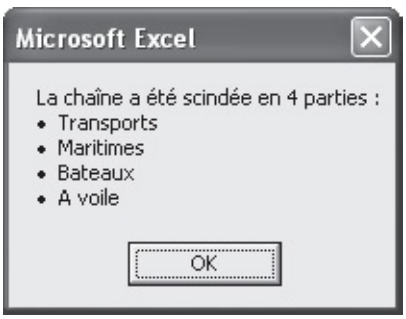


Figure 9-4 – *La fonction InStr() peut servir à scinder une chaîne en fonction d'un séparateur.*

Scinder une chaîne

Dans l'exemple précédent, nous avons scindé une chaîne de caractères en répétant une boucle tant que la comparaison aboutit à un résultat. Cette procédure a servi à illustrer l'utilisation de la fonction `Instr()`. Cependant, une telle démarche revient à écraser une mouche avec un marteau. En effet, VBA propose une fonction dédiée à cette opération : `Split()`.

La fonction `Split()` renvoie les chaînes obtenues sous forme d'un tableau unidimensionnel de base 0 et s'utilise selon la syntaxe suivante :

```
Split(expression, delimiter, limit, compare)
```

où *`expression`* fournit la chaîne de caractères à scinder. Les trois autres arguments sont facultatifs : *`delimiter`* est la chaîne de caractères utilisée comme séparateur (l'espace par défaut), *`limit`* indique le nombre de chaînes à renvoyer (toutes par défaut) et *`compare`* a le même rôle que dans la fonction `Instr()` présentée ci-avant.

Attention

Notez que le résultat retourné par la fonction `Split()` ne peut être stocké que dans une variable de type `Variant`. Si vous tentez de le faire dans une variable tableau, une erreur est générée.

Modifiez la procédure MaMacro() comme suit :

```
 1: Sub MaMacro()
 2:   Dim n As Long
 3:   Dim MonResultat
 4:   Dim Exemple As String
 5:   Dim Message As String
 6:   Exemple = "Transports:Maritimes:Bateaux:A voile"
 7:   MonResultat = Split(Exemple,":")
 8:   For n = 0 To UBound(MonResultat)
 9:     Message = Message & vbCr & Chr(149) & _
         "  " & MonResultat(n)
10:   Next n
11:   MsgBox "La chaîne a été scindée en " & (UBound(MonResultat)+1) & _
       " parties : " & Message
12: End Sub
```

Ligne 3, la variable est maintenant déclarée de type `Variant`, pour recevoir le résultat retourné par la fonction `Split()`. Ligne 7, la fonction `Instr()` précédemment employée dans la boucle est remplacée par `Split()`. Ligne 8, la boucle démarre à zéro, car le tableau retourné par `Split()` est de base zéro. Pour

la même raison, ligne 11, nous avons ajouté 1 à `UBound(MonResultat)` afin de retourner la taille de la variable `MonResultat`.

Rechercher une chaîne dans une variable de matrice

Pour rechercher une chaîne au sein d'une variable de matrice, utilisez la fonction `Filter()`. Elle renvoie un tableau de base zéro dont les espaces de stockage sont les mêmes que ceux de la variable traitée dans lesquels la recherche a abouti. Si la recherche n'a rien trouvé, la fonction renvoie –1. La fonction `Filter()` respecte la syntaxe suivante :

```
Filter(sourcearray, match, include, compare)
```

`sourcearray` représente la matrice à une dimension dans laquelle s'effectue la recherche et *`match`* est la chaîne recherchée. *`include`* et *`compare`* sont facultatifs. Si *`include`* est défini à `True` (valeur par défaut), les contenus des zones de stockage où a abouti la recherche sont retournés. Si *`include`* est défini à `False`, ce sont les zones de stockage dans lesquelles n'a pas abouti la recherche qui sont renvoyées.

Dans l'exemple suivant, on affecte les noms des jours de la semaine à une variable de matrice. On effectue ensuite quatre recherches distinctes au sein de la variable et on en affiche les résultats.

```
 1: Sub RechercherDansUnTableau()
 2:   Dim n As Long
 3:   Dim JoursSemaine(1 To 7)
 4:   Dim MaRecherche1
 5:   Dim MaRecherche2
 6:   Dim MaRecherche3
 7:   Dim MaRecherche4
 8:   Dim Message As String
 9:   For n = 1 To 7
10:     JoursSemaine(n) = WeekdayName(Weekday:=n, abbreviate:=False,
        ➥ firstdayofweek:=vbMonday)
11:   Next n
12:   MaRecherche1 = Filter(sourcearray:=JoursSemaine(), match:="M", include:=True,
       ➥ Compare:=vbBinaryCompare)
13:   MaRecherche2 = Filter(sourcearray:=JoursSemaine(), match:="M",
       ➥ include:=False, Compare:=vbBinaryCompare)
14:   MaRecherche3 = Filter(sourcearray:=JoursSemaine(), match:="M", include:=True,
       ➥ Compare:=vbTextCompare)
15:   MaRecherche4 = Filter(sourcearray:=JoursSemaine(), match:="M",
       ➥ include:=False, Compare:=vbTextCompare)
16:   If Not UBound(MaRecherche1)=-1 Then
17:     Message = "- Résultat de la recherche avec include:=True et
        ➥ Compare:=vbBinaryCompare :" & vbCr
18:     For n = 0 To UBound(MaRecherche1)
19:       Message = Message & " " & MaRecherche1(n) & " /"
20:     Next n
21:   Else
22:     Message = "- Pas de résultat pour la recherche avec include:=True et
```

```
        ➥ Compare:=vbBinaryCompare."
23:   End If
24:   If Not UBound(MaRecherche2)=-1 Then
25:     Message = Message & vbCr & "- Résultat de la recherche avec
        ➥ include:=False et Compare:=vbBinaryCompare " & vbCr
26:     For n = 0 To UBound(MaRecherche2)
27:       Message = Message & " " & MaRecherche2(n) & " / "
28:     Next n
29:   Else
30:     Message = Message & vbCr & "- Pas de résultat pour la recherche avec
        ➥ include:=False et Compare:=vbBinaryCompare."
31:   End If
32:   If Not UBound(MaRecherche3)=-1 Then
33:     Message = Message & vbCr & "- Résultat de la recherche avec include:=True
        ➥ et Compare:=vbTextCompare" & vbCr
34:     For n = 0 To UBound(MaRecherche3)
35:       Message = Message & " " & MaRecherche3(n) & " / "
36:     Next n
37:   Else
38:     Message = Message & vbCr & "- Pas de résultat pour la recherche avec
        ➥ include:=True et Compare:=vbTextCompare."
39:   End If
40:   If Not UBound(MaRecherche4)=-1 Then
41:     Message = Message & vbCr & "- Résultat de la recherche avec
        ➥ include:=False et Compare:=vbTextCompare : " & vbCr
42:     For n = 0 To UBound(MaRecherche4)
43:       Message = Message & " " & MaRecherche4(n) & " / "
44:     Next n
45:   Else
46:     Message = Message & vbCr & "- Pas de résultat pour la recherche avec
        ➥ include:=False et Compare:=vbTextCompare."
47:   End If
48:   MsgBox Message
49: End Sub
```

Lignes 9 à 11, le nom des jours de la semaine est affecté aux sept espaces de stockage de la variable `JoursSemaine`. On fait pour cela appel à la fonction `WeekdayName()` présentée au chapitre précédent. Lignes 12 à 15, on recherche à quatre reprises la chaîne `"M"` dans la variable avec, chaque fois, des valeurs différentes pour les arguments *`include`* et *`compare`*.

Les mêmes instructions sont ensuite appliquées aux quatre résultats de recherche, afin de créer le message à afficher (lignes 16 à 23, 24 à 31, 32 à 39 et 40 à 47). On utilise pour cela une structure `If…Then…Else` afin de savoir si la recherche a abouti. Si c'est le cas, une boucle `For…Next` est utilisée pour concaténer les contenus du tableau ayant reçu le résultat de la fonction.

Enfin, on affiche le message représenté à la figure 9-5.

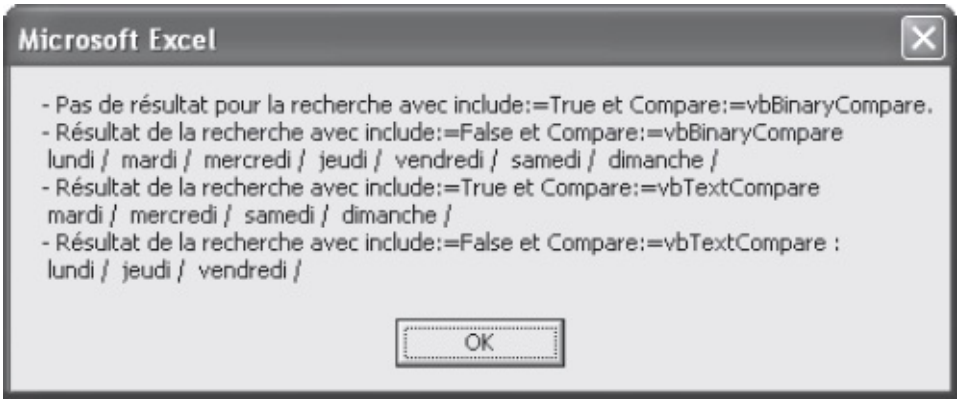

Microsoft Excel
- Pas de résultat pour la recherche avec include:=True et Compare:=vbBinaryCompare.
- Résultat de la recherche avec include:=False et Compare:=vbBinaryCompare
lundi / mardi / mercredi / jeudi / vendredi / samedi / dimanche /
- Résultat de la recherche avec include:=True et Compare:=vbTextCompare
mardi / mercredi / samedi / dimanche /
- Résultat de la recherche avec include:=False et Compare:=vbTextCompare :
lundi / jeudi / vendredi /
OK

Figure 9-5 – *La fonction Filter() permet d'effectuer des recherches inclusives ou exclusives.*

Considérez les résultats de la recherche :

- Dans le premier cas, la recherche ne renvoie aucun résultat, car l'argument `Compare` est défini à `vbBinaryCompare` et s'effectue donc en respectant la casse (M majuscule et non minuscule).
- L'argument `Compare` de la deuxième recherche est également défini à `vbBinaryCompare`. En revanche, la fonction renvoie toutes les valeurs de `JoursSemaine` puisque cette fois-ci l'argument `include` a été défini à `False` (les espaces de stockage renvoyés sont ceux où la recherche n'a pas abouti).
- Pour la troisième recherche, les arguments `include` et `compare` ont respectivement été définis à `True` et `vbTextCompare`. La recherche s'applique donc sans respecter la casse et toutes les zones de stockage contenant la lettre m (minuscule ou majuscule) sont retournées.
- Enfin, la quatrième recherche est effectuée en mode `vbTextCompare`, mais cette fois-ci l'argument `include` a été défini à `False`. Ce sont donc tous les espaces de stockage ne contenant ni M (majuscule) ni m (minuscule) qui sont retournés.

10

Déboguer et gérer les erreurs

Il arrivera immanquablement que des erreurs surviennent lors de l'exécution d'un programme VBA ou que le résultat ne soit pas celui qui était escompté. Vous devrez alors déterminer l'origine de l'erreur et tester de nouveau le programme. VBA dispose pour cela de précieux outils. Ce chapitre vous les présente.

Vous devez tout d'abord distinguer le *débogage* de la *gestion des erreurs*. Le débogage consiste à corriger un programme qui ne fonctionne pas à cause d'un problème lié au code : faute de frappe, syntaxe incorrecte, etc. La *gestion des erreurs* consiste à prévoir les éventuelles erreurs susceptibles de survenir et à y remédier pour que le programme soit aussi fiable que possible, c'est-à-dire s'exécutant correctement dans des contextes différents.

Les étapes et les outils du débogage

Le débogage consiste donc à régler les erreurs directement liées au code et indépendantes de l'environnement dans lequel il s'exécute. Trois types d'erreurs sont susceptibles d'affecter un programme :

- Erreurs de compilation. Elles surviennent lorsque VBA rencontre une instruction qu'il ne reconnaît pas ; par exemple, lorsqu'un mot-clé contient une faute d'orthographe (voir figure 10-1).

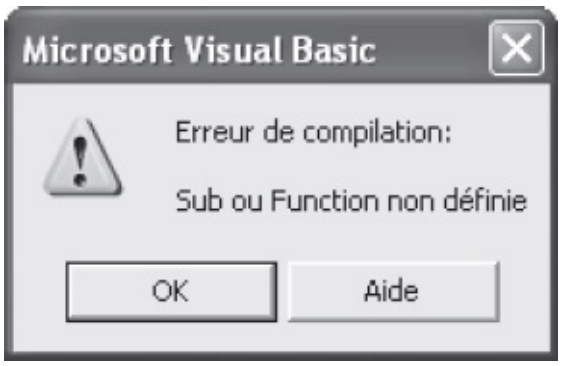


Figure 10-1 – *Les erreurs de compilation sont les plus faciles à repérer.*

- Erreurs d'exécution. Elles surviennent après que la compilation du programme a été réalisée avec succès. Elles peuvent, par exemple, être liées à l'utilisation de données incompatibles. Ce sera le cas si le programme effectue une opération arithmétique sur une variable de chaîne (voir figure 10-2).

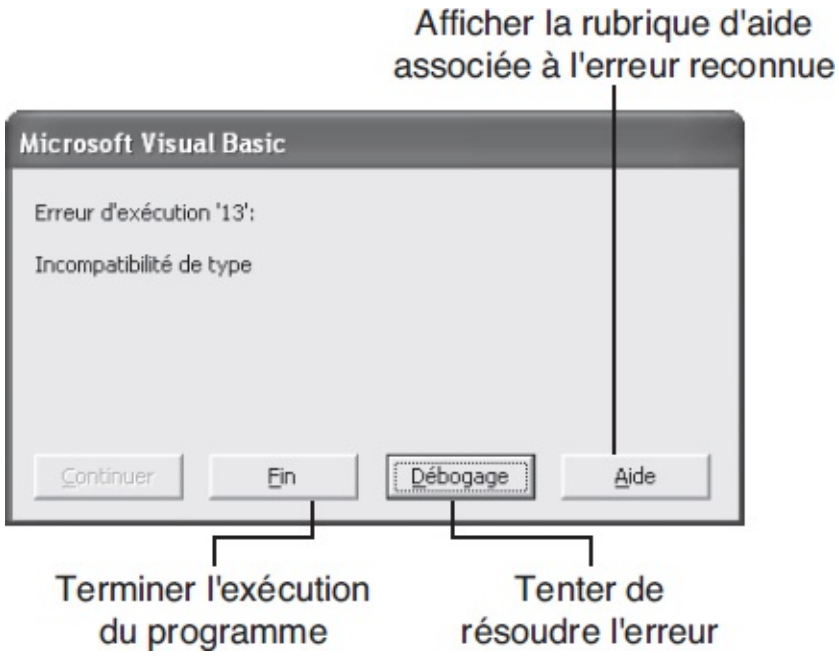


Figure 10-2 – *Une valeur et un message définissent le type d'erreur reconnu par Visual Basic.*

- Erreurs logiques. Elles sont les plus difficiles à redresser. Contrairement aux autres, elles laissent le programme s'exécuter. Le résultat obtenu ne sera pas celui que vous escomptiez.

Les sections suivantes présentent les outils de débogage qui aideront à détecter l'origine de l'erreur.

Conseil

L'activation de l'option Vérification automatique de la syntaxe (voir chapitres 5) de Visual Basic Editor repère les erreurs de syntaxe dès la saisie du code.

Conseil

Il est recommandé de forcer la déclaration explicite des variables à l'aide de l'instruction `Option Explicit`. Vous éviterez ainsi tout risque d'erreur lié à une faute de frappe lors de la saisie d'un nom de

variable. Pour plus d'informations reportez-vous au chapitres 6.

Test du projet

Il est d'usage de tester tout nouveau programme lors du développement. Le premier réflexe consiste à compiler le projet, pour que Visual Basic Editor teste chaque instruction : choisissez la commande Compiler du menu Débogage. Si une erreur est mise en évidence, remédiez-y. Et recommencez jusqu'à ce que la compilation se déroule normalement.

Testez ensuite le projet à partir de Visual Basic Editor : cliquez sur le bouton Exécuter de la barre d'outils Standard.

Conseil

Ne testez pas un programme dont vous n'êtes pas certain sur un document sensible. Il est préférable de travailler sur une copie. Vous serez ainsi assuré de ne pas endommager des données précieuses si le programme ne s'exécute pas correctement, ou s'il produit des résultats erronés.

Si une erreur apparaît à l'exécution, l'instruction coupable est mise en évidence. Remédiez-y. Pour renouveler le test, cliquez sur le bouton Réinitialiser de la barre d'outils Standard ou choisissez la commande Réinitialiser du menu Exécution. La mise en évidence de l'instruction source de l'erreur d'exécution disparaît. Recommencez jusqu'à ce que le programme s'exécute normalement.

Lorsque survient une erreur de compilation ou d'exécution, il est aisé de repérer l'instruction erronée. Si le projet a été compilé à partir de Visual Basic Editor, elle est mise en évidence. Si le programme est exécuté à partir de l'application hôte, la boîte de dialogue de la figure 10-2 s'affiche. Un clic sur le bouton Débogage ouvre Visual Basic Editor (si nécessaire) sur la procédure dans laquelle se trouve l'instruction ayant provoqué l'erreur. Par défaut, celle-ci apparaît en jaune et est signalée par un indicateur de marge (voir figure 10-3).

```
Function ScinderChaine(machaîne As String, separateur As String)
    Dim pos As Long
    Dim mavar
    ReDim mavar(1)
    pos = InStr(machaîne, separateur, 10)
    Do While pos <> 0
        mavar(UBound(mavar)) = Left(machaîne, pos - 1)
        machaîne = Mid(machaîne, pos + Len(separateur))
        pos = InStr(machaîne, separateur)
        ReDim Preserve mavar(UBound(mavar) + 1)
    Loop
    mavar(UBound(mavar)) = machaîne
    ScinderChaine = mavar
End Function
```

Figure 10-3 – *L'instruction ayant provoqué l'erreur est mise en évidence dans Visual Basic Editor.*

Il est plus difficile de repérer les erreurs logiques. Les outils de débobage de Visual Basic Editor sont alors d'un grand secours. Ils sont accessibles *via* le menu Débogage et, pour les plus usités, *via* la barre d'outils Débogage.

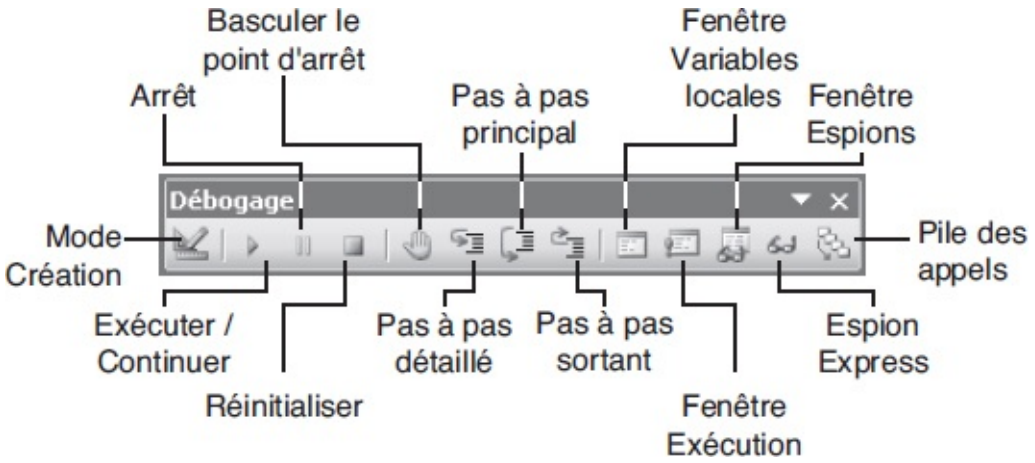


Figure 10-4 – *La barre d'outils Débogage.*

Exécuter pas à pas

Lorsqu'un programme ne provoque pas d'erreur mais ne produit pas le résultat escompté, le premier test consiste à exécuter la procédure pas à pas, afin d'en examiner le déroulement et les conséquences sur le document, instruction après instruction :

1. Placez le curseur dans la procédure à exécuter, puis réduisez Visual Basic Editor afin de visualiser à la fois la fenêtre de l'application hôte et le code à tester.

2. Cliquez sur le bouton Pas à pas détaillé de la barre d'outils, ou choisissez la commande du même nom dans le menu Débogage, ou appuyez sur la touche F8. La première instruction de la procédure est mise en évidence dans la fenêtre Code, tandis que le bouton Arrêt de la barre d'outils Standard apparaît estompé, indiquant que la procédure est interrompue en cours d'exécution.
3. Tapez de nouveau sur la touche F8. L'instruction précédemment mise en évidence dans la fenêtre Code s'exécute, tandis que la suivante est mise en évidence (voir figure 10-5).
4. Répétez l'opération de façon à visualiser l'incidence de chacune des instructions sur le document.

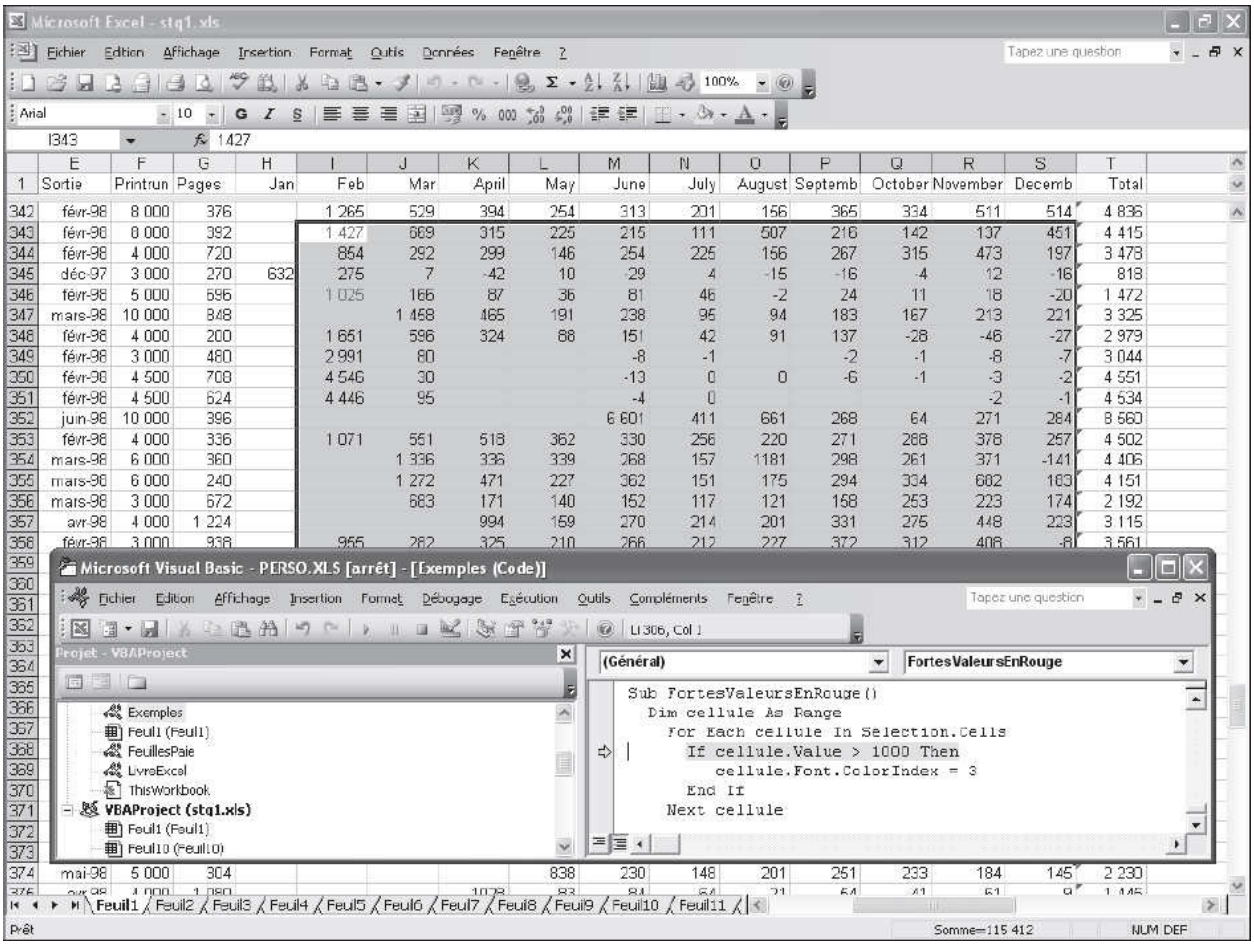


Figure 10-5 – *Exécutez les procédures pas à pas pour visualiser les conséquences de chacune des instructions.*

5. Vous pouvez aussi « lâcher » l'exécution de la procédure en cliquant sur le bouton Continuer de la barre d'outils Standard. Le programme se poursuit alors normalement.

   Vous pouvez aussi exécuter le code procédure par procédure, afin d'étudier la façon dont elles s'appellent dans le programme. Utilisez pour cela la commande Pas à pas principal. La commande Pas à pas sortant exécute tout le code restant dans la procédure en cours et le programme s'interrompt juste après l'instruction d'appel.

La fenêtre Variables locales

Lorsque vous exécutez une procédure pas à pas, elle est en mode Arrêt. Vous pouvez alors visualiser la valeur des variables et des constantes aux différents stades de l'exécution du programme.

Définition

Le mode Arrêt désigne l'état d'une procédure dont l'exécution est interrompue. Cela est dû à une erreur, à l'exécution pas à pas d'une procédure, à la rencontre d'une instruction `End` ou `Stop`, ou à l'interruption manuelle de l'exécution.

Pour visualiser la valeur d'une variable en survolant cette dernière avec la souris (figure 10-6), activez l'option Info-bulles automatiques (Outils > Options).

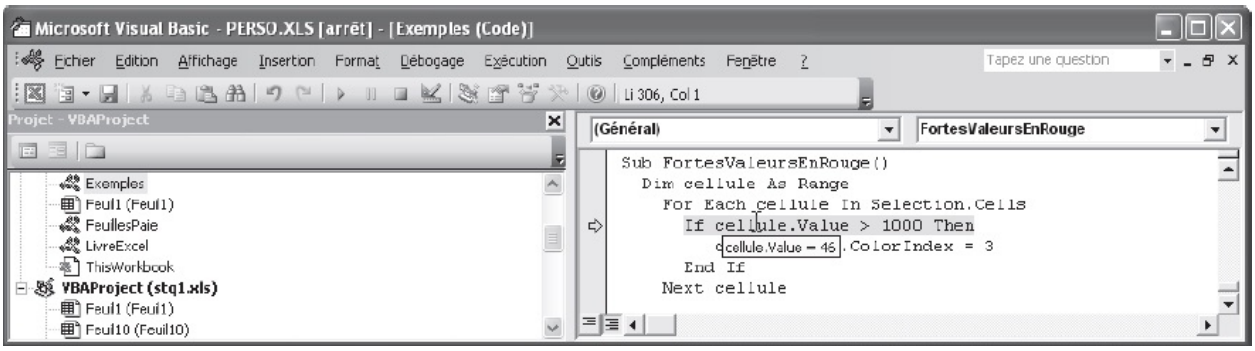


Figure 10-6 – *Les info-bulles automatiques indiquent la valeur des variables à un moment précis de l'exécution d'un programme.*

La fenêtre Variables locales donne des informations précises sur toutes les variables visibles à un moment donné de l'exécution du programme – nom, type et valeur (figure 10-7). Choisissez pour cela Affichage > Variables locales, ou cliquez sur le bouton Variables locales de la barre d'outils Débogage.

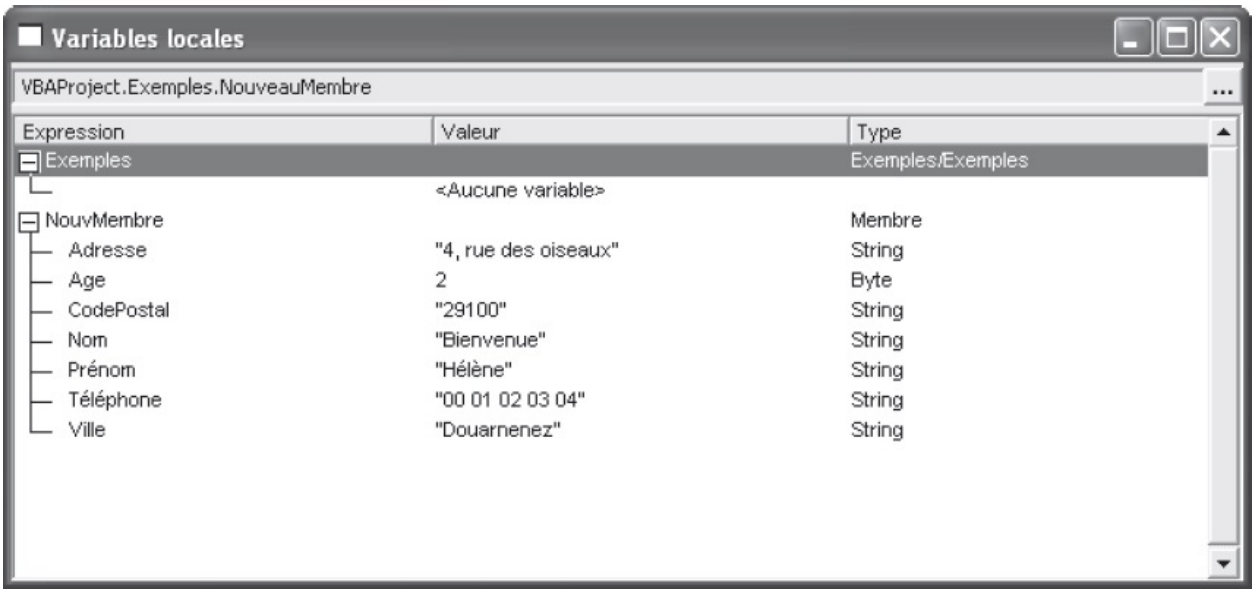


Figure 10-7 – *La fenêtre Variables locales fournit des informations complètes sur les variables.*

Dans le haut de la fenêtre, le nom de la procédure en cours d'exécution s'affiche. Les variables de niveau module apparaissent sous le nom du module, tandis que celles de niveau procédure sont affichées telles quelles.

Si la procédure en cours a été appelée, vous pouvez visualiser les variables des procédures appelantes. Cliquez sur le bouton Pile des appels, sur la procédure dont vous souhaitez visualiser les valeurs, puis sur Afficher. Vous pouvez voir à la figure 10-8 que la procédure `Testdeprocédure2` en cours a été appelée par `Testdeprocédure1`, elle-même appelée par `Testdeprocédure`. Lorsqu'une procédure rend la main à celle qui l'a appelée, elle disparaît de la pile.

Figure 10-8 – *La pile des appels affiche les appels de procédure actifs.*

Vous pouvez modifier les valeurs dans la fenêtre Variables locales, afin de tester le comportement du programme dans d'autres circonstances. Double-cliquez sur la valeur à changer, puis saisissez la valeur voulue. Si cette dernière est incompatible avec le type de la variable, un message d'erreur s'affiche et rien n'est modifié.

Les points d'arrêt

Les points d'arrêt servent à interrompre l'exécution d'un programme sur une instruction précise. Cette possibilité est particulièrement intéressante lorsque vous soupçonnez l'origine d'une erreur. Ainsi, vous exécutez normalement toutes les instructions ne posant pas de problème et vous arrêtez devant une instruction dont vous n'êtes pas sûr. Une fois l'exécution interrompue, vous pouvez la poursuivre pas à pas, examiner la valeur des variables, etc.

Pour définir un point d'arrêt, placez le curseur sur l'instruction voulue et choisissez la commande Basculer le point d'arrêt du menu Débogage, ou cliquez sur le bouton Point d'arrêt de la barre d'outils, ou appuyez sur la touche F9, ou cliquez dans la marge de la fenêtre Code en face de l'instruction voulue. Par défaut, l'instruction apparaît sur un arrière-plan de couleur bordeaux et un indicateur est placé en marge (voir figure 10-9).

Placez plusieurs points d'arrêt dans le code, afin de vérifier l'état des variables ou du document à différents stades, sans avoir à exécuter le programme pas à pas. Pour supprimer un point d'arrêt, procédez de la même façon que pour le placer. Pour supprimer tous ceux d'un module, sélectionnez la commande Effacer tous les points d'arrêt du menu Débogage ou tapez le raccourci clavier correspondant.

Conseil

Lorsque vous quittez Visual Basic Editor, les points d'arrêt ne sont pas enregistrés. Utilisez l'instruction `Stop`, qui entraîne le passage de l'exécution d'une procédure en mode Arrêt.

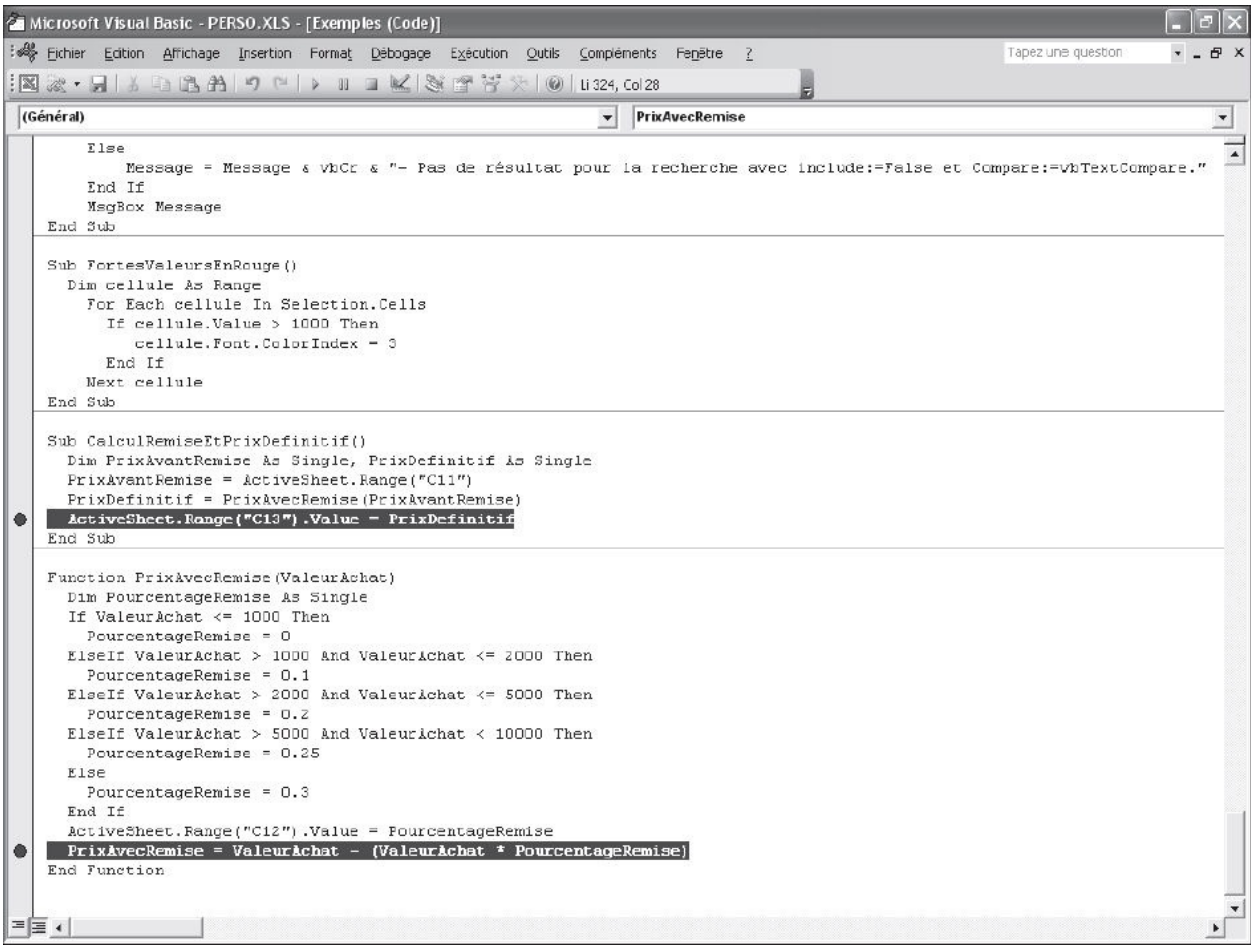


Figure 10-9 – *Les points d'arrêt (ici en gris foncé) définissent des interruptions dans l'exécution du code.*

Modifier l'ordre d'exécution des instructions

En mode Arrêt, vous pouvez à tout moment définir l'instruction suivante à exécuter dans une procédure. Il peut s'agir d'une instruction précédant l'actuelle ou au contraire d'une à venir. Placez le curseur dans l'instruction à exécuter et sélectionnez la commande Définir l'instruction suivante du menu Débogage, ou faites glisser l'indicateur de marge vers l'instruction voulue. Le code intermédiaire est alors ignoré et la procédure se poursuit à partir de l'instruction définie.

Ainsi, vous ignorez une série d'instructions ne vous intéressant pas dans le cadre du débogage, ou au contraire évitez celles qui provoquent une erreur, pour étudier le comportement du programme dans d'autres circonstances. Vous pouvez alors utiliser la fenêtre Exécution pour lancer des instructions n'apparaissant pas dans votre code, mais que vous envisagez de substituer aux instructions ignorées.

Info

L'instruction suivante ne peut être définie qu'à l'intérieur de la procédure en cours.

La fenêtre Exécution

En mode Arrêt, la fenêtre Exécution sert à lancer tout type d'instruction qui ne se trouve pas dans le code du programme. Il suffit d'y écrire l'instruction et d'appuyer sur Entrée ; elle s'exécute alors comme si elle faisait partie intégrante du code.

Choisissez la commande Fenêtre Exécution, ou cliquez sur le bouton Fenêtre Exécution de la barre d'outils Débogage, ou encore saisissez le raccourci clavier Ctrl+G. À la figure 10-10, la valeur indiquée par l'utilisateur et affectée à la variable `DateEcheance` a provoqué une erreur. Nous avons placé un point d'arrêt sur l'instruction incriminée, afin d'y revenir par la suite. Nous avons utilisé la commande Définir l'instruction suivante du menu Débogage pour poursuivre l'exécution du code à la ligne suivante. Enfin, nous avons redéfini la valeur de `DateEcheance` à "01/04/2017" et nous avons relancé l'instruction ayant provoqué l'erreur dans la fenêtre Exécution.

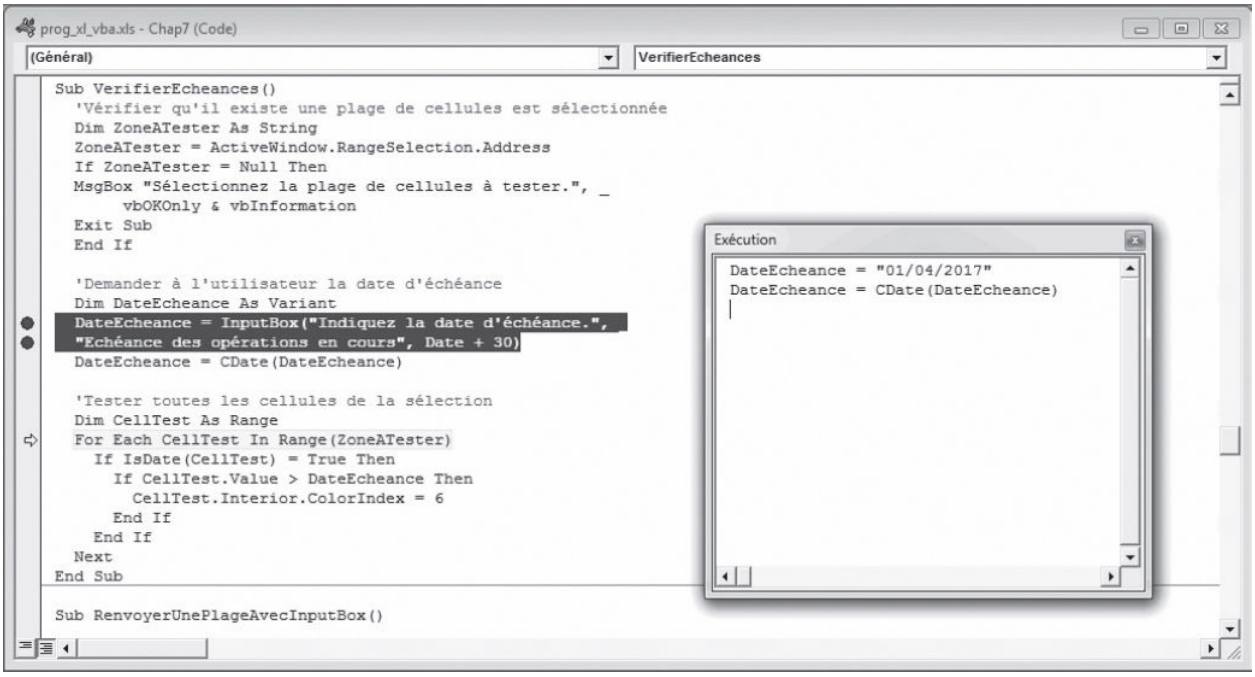


Figure 10-10 – *La fenêtre Exécution permet d'exécuter une instruction comme si elle faisait partie intégrante du code.*

Les espions

Les espions aident à suivre les valeurs de variables ou de toute expression renvoyant une valeur dans un contexte déterminé. Procédez ainsi :

1. Choisissez la commande Ajouter un espion du menu Débogage (voir figure 10-11).
2. Saisissez l'expression dont vous souhaitez espionner la valeur.
3. Déterminez le contexte dans lequel l'expression sera espionnée. Par défaut, les zones Procédure et Module affichent respectivement la procédure et le module en cours. Vous pouvez choisir d'espionner la valeur d'une expression dans une procédure d'un module, dans toutes les procédures d'un module, ou encore dans tous les modules. Il suffit, en général, de limiter la portée des espions à celle des expressions espionnées.

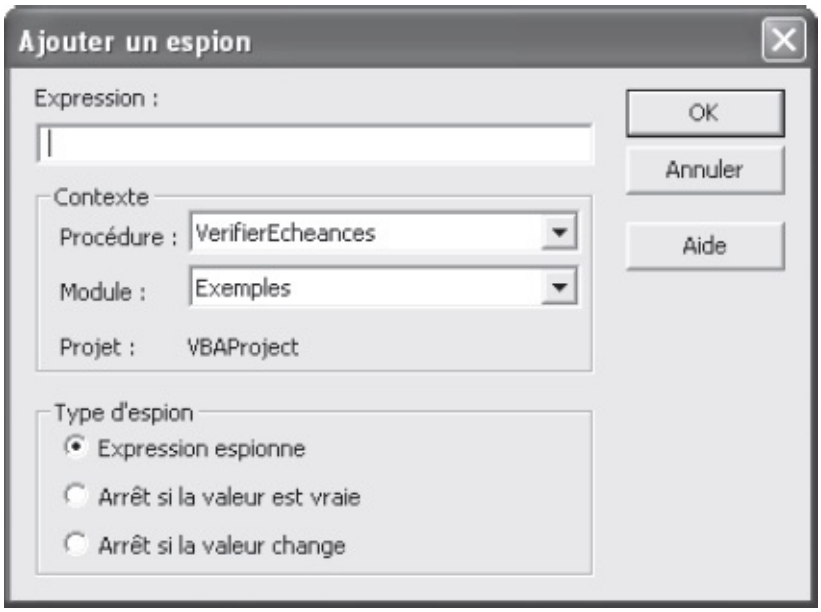


Figure 10-11 – *La boîte de dialogue Ajouter un espion.*

4. Dans la zone Type d'espion, choisissez l'une des trois options disponibles :
   - Expression espionne. En mode Arrêt, la valeur en cours de l'expression s'affiche dans la fenêtre Espions.
   - Arrêt si la valeur est vraie. La procédure passe en mode Arrêt si la valeur de l'expression est définie à `True`.
   - Arrêt si la valeur change. L'exécution du code s'interrompt si la valeur de l'expression change.
5. Cliquez sur OK pour valider.

Vous pouvez aussi sélectionner la variable ou l'expression voulue et la faire glisser dans la fenêtre Espions.

Pour afficher la fenêtre, sélectionnez la commande Fenêtre Espions du menu Affichage, ou cliquez sur le bouton Fenêtre Espions de la barre d'outils Débogage. Sept espions ont été placés dans la fenêtre représentée à la figure 10-12.

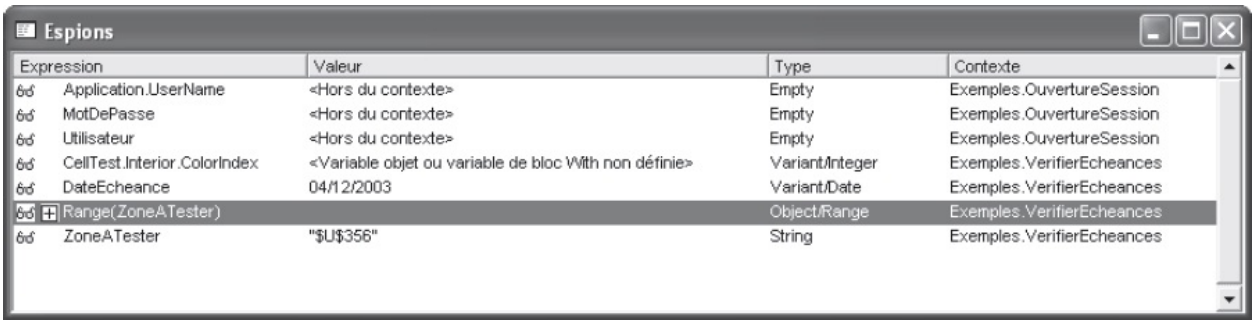


Figure 10-12 – *Placez des espions dans votre code…*

La fenêtre Espions contient quatre champs :

- Expression affiche l'expression espionnée. Il peut s'agir d'un nom de variable ou de toute expression renvoyant une valeur.
- Valeur affiche la valeur actuelle de l'expression espionnée en mode Arrêt. Dans le cas d'une expression de niveau procédure, cette zone affiche <Hors du contexte> si la procédure ne fait pas partie des appels de procédure actifs (elle n'apparaît alors pas dans la pile des appels).

  Vous pouvez modifier la valeur d'une expression dans la fenêtre Espions, afin de tester le comportement du programme dans d'autres circonstances. Double-cliquez sur la valeur, puis changez-la. Si la valeur choisie est incompatible avec l'expression, un message d'erreur s'affiche et rien ne change.
- Type affiche le type de la variable ou de la valeur renvoyée par l'expression. Si l'instruction en cours est hors du contexte d'espionnage, cette zone affiche `Empty` ou `Variant/Empty`.
- Contexte affiche le contexte défini pour l'expression espionne. À la figure 10-12, vous constatez que le premier espion est défini pour toutes les procédures de Module1, tandis que les autres sont limités à une procédure spécifique du module.

Pour supprimer un espion, sélectionnez-le et appuyez sur Suppr, ou cliquez-droit dans la fenêtre Espions et choisissez la commande Supprimer un espion du menu contextuel.

Si vous souhaitez connaître la valeur d'une expression pour laquelle vous n'avez pas placé d'espion, faites appel à l'Espion express : sélectionnez l'expression voulue, puis choisissez la commande Expression express du menu Débogage, ou cliquez sur le bouton Espion express de la barre d'outils

Débogage, ou encore utilisez Maj+F9. La boîte de dialogue vous renseigne alors sur le contexte de l'expression sélectionnée et sur sa valeur (voir figure 10-13). Pour intégrer cette expression à la fenêtre Espions, cliquez sur Ajouter.

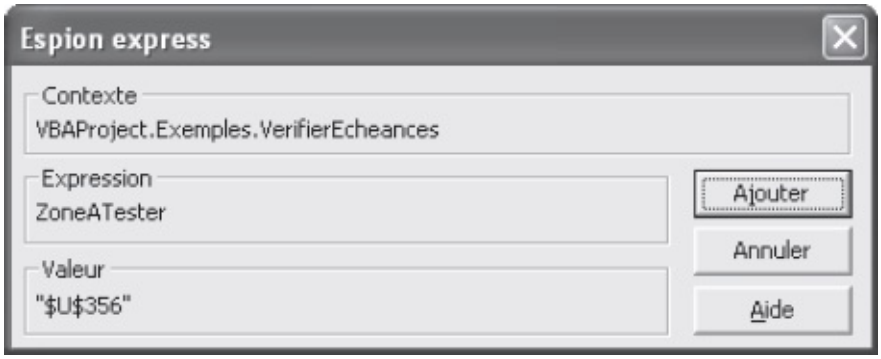


Figure 10-13 – *L'Espion express : toujours prêt.*

La pile des appels

La boîte de dialogue Pile des appels recense toutes les procédures en cours d'exécution, selon leur ordre d'appel, la dernière appelée étant en haut de la liste.

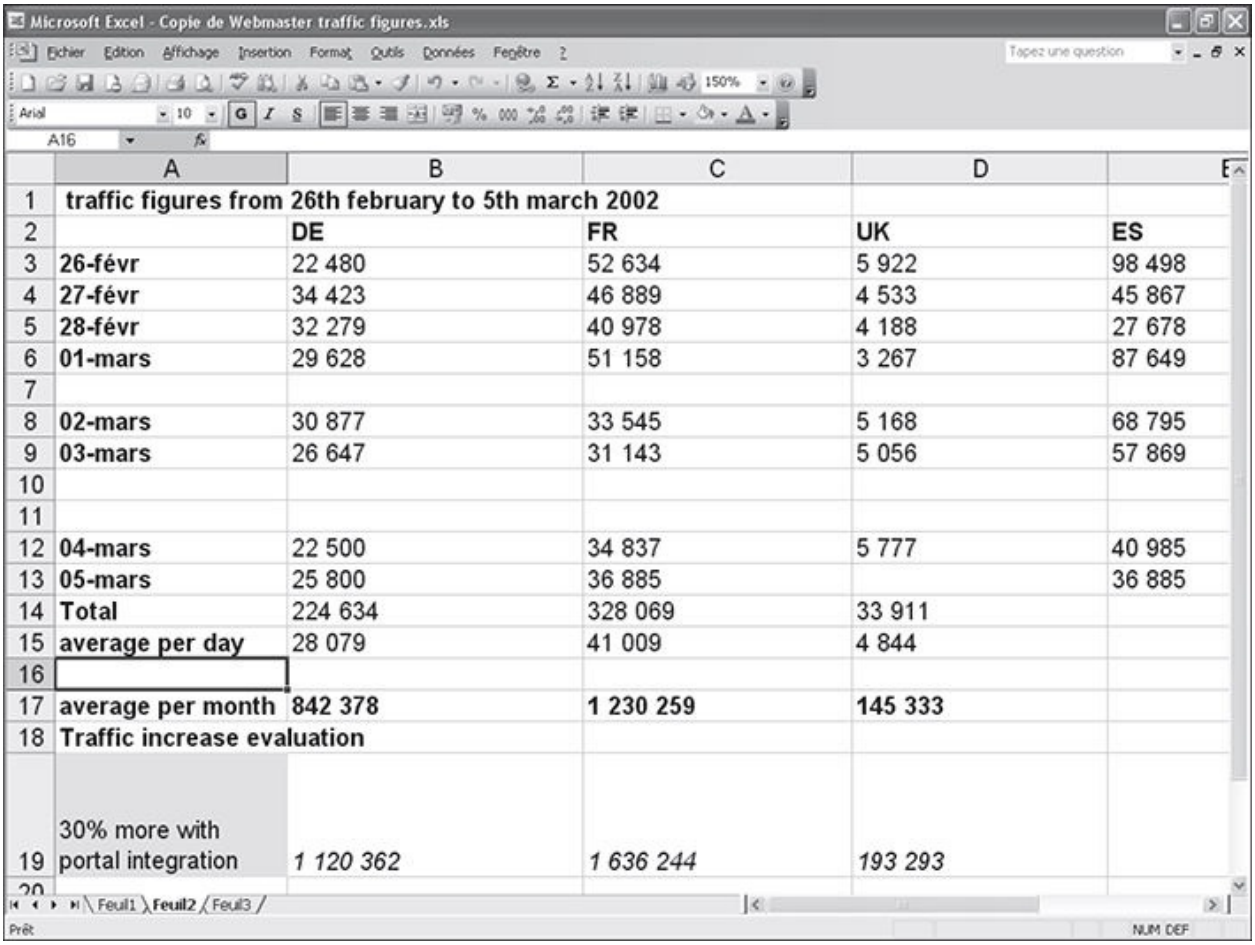
Il est intéressant de visualiser la pile des appels lors du débogage d'un programme VBA. Vous avez ainsi une idée précise des procédures en cours d'exécution et des appels successifs. Choisissez la commande Pile des appels du menu Affichage, ou cliquez sur le bouton Pile des appels de la barre d'outils Débogage, ou encore tapez le raccourci clavier Ctrl+L.

Sur la figure 10-14, vous voyez que la procédure en cours d'exécution est `Procédure3`, qui a été appelée par `Procédure2`, elle-même appelée par `AppelsDeProcédures`.

Figure 10-14 – *La boîte de dialogue Pile des appels.*

Exemple de débogage

Nous allons créer ici un programme que nous déboguerons jusqu'au moment où nous atteindrons une version fiable. Nous supposons que nous possédons un classeur Excel contenant de nombreuses données, mais dans lequel certaines lignes sont vides. Nous décidons donc d'écrire une macro VBA pour supprimer ces dernières. Nous supposons ici que, lorsqu'une cellule de la colonne A ne contient pas de données, la ligne est vide et doit être supprimée. Ainsi, les lignes 7, 10, 11 et 16 du classeur représenté à la figure 10-15 doivent être supprimées.



	A	B	C	D	E
1	traffic figures from 26th february to 5th march 2002				
2		DE	FR	UK	ES
3	26-févr	22 480	52 634	5 922	98 498
4	27-févr	34 423	46 889	4 533	45 867
5	28-févr	32 279	40 978	4 188	27 678
6	01-mars	29 628	51 158	3 267	87 649
7					
8	02-mars	30 877	33 545	5 168	68 795
9	03-mars	26 647	31 143	5 056	57 869
10					
11					
12	04-mars	22 500	34 837	5 777	40 985
13	05-mars	25 800	36 885		36 885
14	Total	224 634	328 069	33 911	
15	average per day	28 079	41 009	4 844	
16					
17	average per month	842 378	1 230 259	145 333	
18	Traffic increase evaluation				
19	30% more with portal integration	*1 120 362*	*1 636 244*	*193 293*	
20					

Figure 10-15 – *Le programme devra supprimer les lignes ne contenant pas de données.*

Commençons logiquement par définir la zone à traiter (de A1 à la dernière

cellule de la colonne A contenant des données) et l'affecter à une variable objet. Nous utiliserons ensuite une structure `For Each…Next` pour vérifier la valeur de chacune des cellules de la zone préalablement définie et supprimer la ligne correspondante chaque fois qu'il n'y a pas de données. Le programme correspondant se présente ainsi :

```
 1: Sub SupprLignesVidesBoguée()
 2:   'Définir la plage à tester
 3:   'On commence par définir la dernière cellule
 4:   'de la colonne A contenant des données
 5:   Dim MaPlage As Range
 6:   Set MaPlage = Range("A1")
 7:   While Range(MaPlage.End(xlDown).Address(rowabsolute:=False,
      ➥ columnabsolute:=False)).Value<>""
 8:     Set MaPlage = Range(MaPlage.End(xlDown).Address(rowabsolute:=False,
        ➥ columnabsolute:=False))
 9:   Wend
10:   Set MaPlage = Range("A1:" & MaPlage.Address(rowabsolute:=False,
      ➥ columnabsolute:=False))
11:
12:   'Supprimer les lignes ne contenant pas de données
13:   Dim Cellule As Range
14:   For Each Cellule In MaPlage
15:     If Cellule.Value="" Then
16:       Rows(Cellule.Row).Delete
17:     End If
18:   Next Cellule
19: End Sub
```

Lignes 5 à 10, la plage de cellules à tester est définie et affectée à la variable `MaPlage` de type `Range`. On commence par déclarer cette dernière et lui affecter la cellule A1 (lignes 5 et 6). Lignes 7 à 9, une structure `While…Wend` sert à définir la dernière cellule de la colonne A contenant des données. `MaPlage` se voit affecter la cellule renvoyée par la propriété `End` (ligne 8), tant que celle-ci n'est pas vide (ligne 7). Lorsque la condition n'est plus vérifiée, la cellule affectée à `MaPlage` est la dernière contenant des données. La boucle `While…Wend` prend alors fin. Ligne 10, `MaPlage` est redéfinie de A1 à la dernière cellule identifiée.

Lignes 14 à 18, une instruction `For Each…Next` est utilisée pour tester chacune des cellules de `MaPlage`. Une instruction conditionnelle vérifie si la cellule est vide, auquel cas la ligne correspondante est supprimée (ligne 16).

Recherche du bogue

Exécutez le programme sur un classeur Excel contenant des lignes vides. Vous constatez que, s'il existe deux lignes vides consécutives, la deuxième n'est pas supprimée.

Pour comprendre d'où vient le problème, procédez comme suit :

1. Le problème ne vient visiblement pas de la première partie du programme, puisque la plage de cellules à tester est correctement définie. Placez donc un point d'arrêt devant l'instruction de la ligne 14, qui semble provoquer l'erreur.
2. Exécutez ensuite la macro. Lorsque la ligne 14 est atteinte, le programme s'interrompt et Visual Basic Editor passe au premier plan. L'instruction contenant le point d'arrêt est en surbrillance et le bouton Arrêt de la barre d'outils Standard est estompé, indiquant le mode Arrêt.
3. Réduisez la fenêtre de Visual Basic Editor de façon à voir le classeur Excel au second plan. Poursuivez ensuite l'exécution de la macro pas à pas (touche F8).
4. Observez attentivement l'effet de chaque instruction sur le classeur, ainsi que les valeurs que prennent les différentes variables et expressions du programme. Placez pour cela le curseur au-dessus des expressions `Cellule.Value` et `Cellule.Row` pour afficher les info-bulles automatiques. Soyez particulièrement attentif lorsqu'une cellule vide est testée.

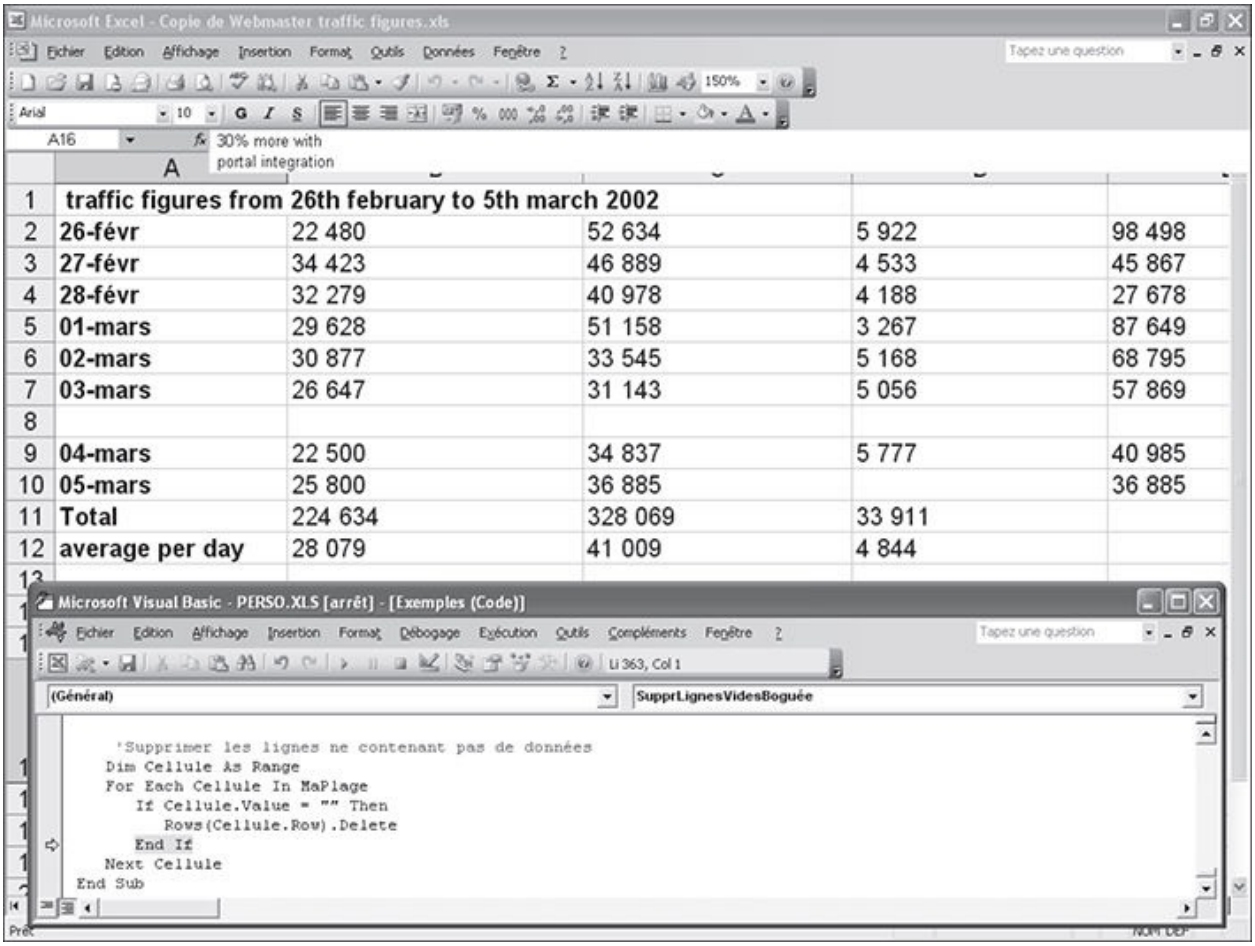

Microsoft Excel - Copie de Webmaster traffic figures.xls
Fichier Edition Affichage Insertion Format Outils Données Fenêtre ?
Tapez une question
A16
30% more with portal integration
traffic figures from 26th february to 5th march 2002
26-févr 22 480 52 634 5 922 98 498
27-févr 34 423 46 889 4 533 45 867
28-févr 32 279 40 978 4 188 27 678
01-mars 29 628 51 158 3 267 87 649
02-mars 30 877 33 545 5 168 68 795
03-mars 26 647 31 143 5 056 57 869
04-mars 22 500 34 837 5 777 40 985
05-mars 25 800 36 885 36 885
Total 224 634 328 069 33 911
average per day 28 079 41 009 4 844
Microsoft Visual Basic - PERSO.XLS [arrêt] - [Exemples (Code)]
Fichier Edition Affichage Insertion Format Débogage Exécution Outils Compléments Fenêtre ?
Li 363, Col 1
(Général)
SupprLignesVidesBoguée
'Supprimer les lignes ne contenant pas de données
Dim Cellule As Range
For Each Cellule In MaPlage
If Cellule.Value = "" Then
Rows(Cellule.Row).Delete
End If
Next Cellule
End Sub

Figure 10-16 – *Exécutez la macro pas à pas et observez attentivement ses effets sur la feuille Excel, ainsi que les valeurs que prennent les variables.*

5. Vous devriez vous rendre compte que, lorsque la ligne 10 est supprimée, la ligne 11 prend sa place et que la cellule suivante testée est A11. Le contenu de la cellule précédemment en A11 ne sera donc pas traité, puisqu'il est passé en A10.

La boucle `For Each…Next` traite les cellules de MaPlage une à une. Or, lorsqu'une ligne est supprimée, la suivante prend sa place et elle est donc ignorée par la procédure.

Résolution du bogue

Plusieurs solutions permettent de régler ce problème.

Vous pouvez intégrer une instruction contrôlant de nouveau le contenu de la cellule lorsqu'une ligne est supprimée :

```
If Cellule.Value="" Then
   Dim LigneSuppr As Long
   LigneSuppr = Cellule.Row
   Rows(Cellule.Row).Delete
   If Range("A" & LigneSuppr)="" Then
      Rows(Range("A" & LigneSuppr).Row).Delete
   End If
End If
```

Avant de supprimer une ligne, son numéro est stocké dans `LigneSuppr`. La ligne est ensuite supprimée. Une instruction conditionnelle imbriquée la supprime à nouveau si la cellule de la colonne A correspondante est encore vide. Le problème est ainsi réglé si deux lignes consécutives sont vides, mais pas s'il en existe plus de deux.

Une autre solution consiste à utiliser une structure `For…Next` à la place de `For Each…Next`. Vous définirez un déroulement de la boucle commençant par la dernière cellule plutôt que par la première. Le programme se présente alors ainsi :

```
Sub SupprLignesVidesVersion2()
   Dim MaPlage As Range
   Set MaPlage = Range("A1")
   While Range(MaPlage.End(xlDown).Address(rowabsolute:=False,
     ➥ columnabsolute:=False)).Value<>""
     Set MaPlage = Range(MaPlage.End(xlDown).Address(rowabsolute:=False,
       ➥ columnabsolute:=False))
   Wend


   Dim DerLigne As Long
   DerLigne = MaPlage.Row
   'Supprimer les lignes ne contenant pas de données
```

```
   Dim Compteur As Long
   For Compteur = DerLigne To 1 Step -1
     If Range("A" & Compteur).Value="" Then
       Rows(Range("A" & Compteur).Row).Delete
     End If
   Next Compteur
End Sub
```

Une troisième solution consistera à stocker la liste des cellules vides dans une variable de matrice et à ne procéder à la suppression des lignes vides qu'une fois la liste définie. On devra là aussi partir de la dernière jusqu'à atteindre la première.

```
 1: Sub SupprLignesVidesVersion3()
 2:   'Création d'une variable de matrice redimensionnable
 3:   Dim MonTableau()
 4:
 5:   'Définir la plage à tester
 6:   Dim MaPlage As Range
 7:   Set MaPlage = Range("A1")
 8:   While Range(MaPlage.End(xlDown).Address(rowabsolute:=False,
       ➥ columnabsolute:=False)).Value<>""
 9:     Set MaPlage = Range(MaPlage.End(xlDown).Address(rowabsolute:=False,
        ➥ columnabsolute:=False))
10:   Wend
11:   Set MaPlage = Range("A1:" & MaPlage.Address(rowabsolute:=False,
       ➥ columnabsolute:=False))
12:
13:   'On stocke dans la variable de matrice les valeurs
14:   'de toutes les lignes à supprimer
15:   Dim Cellule As Range
16:   For Each Cellule In MaPlage
17:     If Cellule.Value="" Then
18:       'Une erreur sera générée à l'appel de la fonction
19:       'UBound si le tableau n'a encore reçu aucune valeur
20:       On Error Resume Next
21:         ReDim Preserve MonTableau(UBound(MonTableau) + 1)
22:       'L'instruction conditionnelle suivante gère cette erreur
23:       If Err.Number=9 Then
24:         ReDim MonTableau(1)
25:         Err.Clear
26:       End If
27:       MonTableau(UBound(MonTableau)) = Cellule.Row
28:     End If
29:   Next Cellule
30:
31:   'Suppression des lignes vides
32:   Dim Compteur As Single
33:   For Compteur = UBound(MonTableau) To LBound(MonTableau) Step -1
34:     Rows(MonTableau(Compteur)).Delete
35:   Next Compteur
36: End Sub
```

L'instruction `Option Base 1` a été placée dans la zone de déclaration du module, de façon que la première valeur d'index des tableaux soit 1 et non 0.

Ligne 3, un tableau redimensionnable est déclaré. Il servira à stocker les numéros des lignes à supprimer. Lignes 6 à 11, la zone à traiter est définie et affectée à `MaPlage`.

Lignes 15 à 29, les numéros des lignes à supprimer sont définis et stockés dans `MonTableau`. La variable `Cellule` de type `Range` est tout d'abord déclarée. Une structure `For Each…Next` sert ensuite à tester l'ensemble des cellules de `MaPlage` (lignes 16 à 29). À chaque passage de la boucle, une structure conditionnelle (lignes 17 à 28) évalue le contenu de la cellule testée et stocke le numéro de la ligne dans `MonTableau` si la cellule est vide. La variable est tout d'abord redimensionnée (ligne 21). On se sert pour cela de la fonction `UBound` qui renvoie la taille du tableau, à laquelle on ajoute 1. Ligne 27, le dernier espace de stockage de `MonTableau` reçoit le numéro de la ligne à supprimer.

Rappel

Notez l'utilisation du mot-clé `Preserve` lors du redimensionnement du tableau (ligne 21). Celui-ci est indispensable pour que le tableau ne soit pas réinitialisé.

Lignes 20 à 26, un gestionnaire d'erreur a été mis en place. En effet, la première fois que la fonction `UBound` est utilisée (ligne 21), une erreur est générée. L'instruction `Resume Next` de la ligne 20 force le passage à l'instruction suivante en cas d'erreur. Ligne 23, on teste la valeur de la propriété `Number` de l'objet `Err` pour vérifier si une erreur a été générée. Si tel est le cas, `MonTableau` est dimensionné avec un seul espace de stockage et l'objet `Err` est réinitialisé à l'aide de la méthode `Clear`.

Conseil

Pour connaître le numéro d'une erreur (la valeur de la propriété `Number` de l'objet `Err`), provoquez-la volontairement et relevez le numéro indiqué dans la boîte de dialogue Visual Basic affichée.

Lignes 32 à 35, les lignes vides sont supprimées. On utilise pour cela une boucle `For…Next` dont le compteur commence à la valeur d'index la plus importante de `MonTableau` pour atteindre la valeur la plus basse, en décrémentant de 1 à chaque passage. Dans la boucle, l'instruction de la ligne 34 supprime la ligne dont le numéro correspond à la valeur stockée dans l'espace d'index *`Compteur`* de `MonTableau`.

Info

L'intérêt de cette version du programme est de stocker les lignes supprimées dans une variable. Placez les instructions suivantes en fin de programme pour afficher le nombre et la liste des lignes supprimées :

```
Dim Message As String
   Message = UBound(MonTableau) & " lignes ont été supprimées :"
   For Compteur = 1 to UBound(MonTableau)
      Message = Message & vbCr & MonTableau(Compteur)
```

```
    Next Compteur
    MsgBox Message, vbOKOnly + vbInformation
```

En l'état, le programme lancera encore une erreur à la ligne 33 si aucune ligne n'est à supprimer. Corrigez ce bogue selon la méthode de votre choix : soit en encadrant les instructions par une structure conditionnelle qui vérifiera que `MonTableau` n'est pas vide, soit par la mise en place d'un gestionnaire d'erreur.

Figure 10-17 – *Cette version du programme indique les lignes supprimées.*

Une telle méthode se révélera particulièrement intéressante si vous créez une procédure supprimant des lignes dont vous souhaitez récupérer les informations dans un autre classeur Excel. Vous utiliserez alors une variable de matrice dynamique à deux dimensions : l'une correspondant aux lignes du classeur, l'autre aux colonnes. La taille de la première dimension (le nombre de lignes à supprimer ou le nombre de colonnes du classeur contenant des données à conserver) devra être définie avant de stocker les données dans la variable – seule la dernière dimension pouvant être redéfinie en conservant les valeurs de la variable.

Gestion des erreurs et des exceptions

Un programme VBA peut s'exécuter correctement dans la plupart des cas et provoquer des erreurs d'exécution dans des contextes spécifiques. Une erreur sera, par exemple, lancée si l'utilisateur n'entre pas le type d'information attendu dans une boîte de dialogue ou si le format d'une cellule ne correspond pas au type de données qu'un programme tente d'exploiter, ou encore si un programme tente de modifier un classeur Excel en cours d'utilisation. Nombre d'erreurs de ce type, liées à un code écrit pour un contexte particulier, sont susceptibles d'affecter une macro. Il est important de les prévoir et de mettre

en place un *gestionnaire d'erreur*, afin que la procédure les contourne et s'exécute normalement.

Définition

Un gestionnaire d'erreur est un ensemble d'instructions qui est censé permettre la poursuite de l'exécution d'une procédure. En général, il est appelé par un détecteur d'erreur.

Pour détecter une éventuelle erreur, placez une instruction `On Error` devant l'instruction susceptible d'en lancer une :

- **`On Error Resume Next`** ignore l'instruction ayant provoqué une erreur et passe à la suivante.
- **`On Error GoTo` *`étiquette`*** appelle un gestionnaire d'erreur repéré par l'étiquette.

Une technique courante consiste à placer le gestionnaire d'erreur à la fin de la procédure et à glisser une instruction `Exit` avant l'étiquette le délimitant :

```
Sub MaProcédure()
  …
  On Error GoTo GestionnaireErreur
  Instruction susceptible de provoquer une erreur
  …
  Exit Sub
GestionnaireErreur:
  Instructions de gestion des erreurs
  Resume
End Sub
```

Ainsi, dès qu'une erreur est détectée, le gestionnaire est appelé. L'instruction `Resume` ne peut être placée que dans un tel gestionnaire : elle rend la main à l'instruction ayant provoqué une erreur, qui s'exécute de nouveau. Si aucune erreur ne se reproduit, le programme se déroule normalement jusqu'à l'instruction `Exit Sub`. La procédure s'achève alors, sans que le gestionnaire d'erreur ait été exécuté.

Lorsqu'une erreur survient, son type est affecté à la propriété `Number` de l'objet `Err`. Après gestion de l'erreur, il est important de réinitialiser cette propriété afin que les éventuelles autres erreurs puissent être détectées : `Err.Number = 0`.

Exemple de gestion d'erreur

La procédure suivante – étudiée au chapitres 7 – provoquera une erreur si l'utilisateur n'entre pas une date avec un format valide, dans la boîte de dialogue affichée par la fonction `InputBox` (voir figure 10-18).

```
Sub VerifierEcheances()
  'Vérifier qu'une plage de cellules est sélectionnée
```

```
  Dim ZoneATester As String
  ZoneATester = ActiveWindow.RangeSelection.Address
  If ZoneATester=Null Then
    MsgBox "Sélectionnez la plage de cellules à tester.", _
      vbOKOnly + vbInformation
    Exit Sub
  End If


  'Demander la date d'échéance à l'utilisateur
  Dim DateEcheance As Variant
  DateEcheance = InputBox("Indiquez la date d'échéance.", _
    "Echéance des opérations en cours", Date + 30)
  If DateEcheance="" Then
    Exit Sub
  End if
  DateEcheance = CDate(DateEcheance)
  'Tester toutes les cellules de la sélection
  Dim CellTest As Range
  For Each CellTest In Range(ZoneATester)
    If IsDate(CellTest)=True Then
      If CellTest.Value>DateEcheance Then
        CellTest.Interior.ColorIndex = 6
      End If
    End If
  Next
End Sub
```

Figure 10-18 – *Cette erreur peut être gérée par le programme.*

Modifiez-la comme suit : un gestionnaire d'erreur affiche une nouvelle boîte de dialogue dans laquelle l'utilisateur est invité à entrer une date dans un format valide. L'instruction ayant provoqué l'erreur est alors répétée et le programme se déroule normalement, jusqu'à l'instruction `Exit Sub` qui entraîne la sortie de la procédure. Si l'utilisateur clique sur le bouton Annuler, la variable `DateEcheance` renverra `Empty` et l'instruction `Exit Sub` entraînera la sortie du programme.

```
Sub VerifierEcheances()
  Dim ZoneATester As String
  ZoneATester = ActiveWindow.RangeSelection.Address
```

```
  If ZoneATester=Null Then
    MsgBox "Sélectionnez la plage de cellules à tester.", _
      vbOKOnly + vbInformation
    Exit Sub
  End If


  Dim DateEcheance As Variant
  DateEcheance = InputBox("Indiquez la date d'échéance.", _
    "Echéance des opérations en cours", Date + 30)
  If DateEcheance="" Then
    Exit Sub
  End if
  On Error GoTo GestionnaireErreur
  DateEcheance = CDate(DateEcheance)


  Dim CellTest As Range
  For Each CellTest In Range(ZoneATester)
    If IsDate(CellTest)=True Then
      If CellTest.Value>DateEcheance Then
        CellTest.Interior.ColorIndex = 6
      End If
    End If
  Next


  Exit Sub


GestionnaireErreur:
  Err.Number = 0
  DateEcheance = InputBox("Format de date non valide." & _
    Chr(10) & "Entrez une date au format jj/mm/aa", "Erreur gérée", _
    "jj/mm/aa")
  If DateEcheance=Empty Then Exit Sub
  Resume
End Sub
```

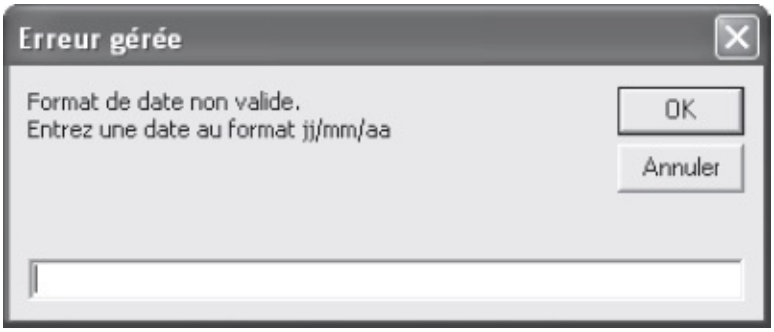


Figure 10-19 – *L'erreur est détectée et l'utilisateur est invité à entrer une information valide ou à annuler l'opération.*

11

Intégrer des applications VBA dans l'interface d'Excel

Il est simple d'améliorer l'accessibilité d'une macro en lui affectant un raccourci clavier, une ligne de commande dans un menu, un bouton de barre d'outils ou tout objet figurant sur une feuille de calcul.

Si vous n'utilisez que rarement une macro, contentez-vous de l'exécuter par la boîte de dialogue Macros. Cependant, sa nature et sa fréquence d'utilisation justifient souvent un accès plus rapide. Une macrocommande ne présente qu'un intérêt limité si elle ne peut être exécutée rapidement, par un raccourci clavier ou une icône (ou les deux). Il sera en revanche préférable d'affecter une ligne de commande à un programme aux conséquences plus larges et d'une utilisation moins fréquente.

Affecter une macro à un raccourci clavier

Si vous n'avez pas affecté de raccourci clavier à une macro au moment de sa création, il est très simple de le faire par la suite :

1. Cliquez sur le bouton Macros de l'onglet Développeur.
2. Dans la boîte de dialogue, sélectionnez la macro voulue puis cliquez sur le bouton Options (voir figure 11-1).
3. Indiquez le raccourci de votre choix en saisissant une lettre dans la zone conçue à cet effet. Vous pouvez aussi ajouter une description, si vous avez omis de le faire lors de la création.
4. Cliquez sur OK pour valider les modifications.

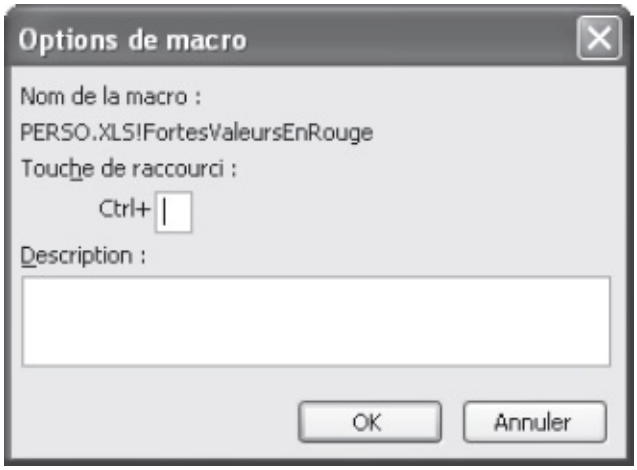


Figure 11-1 – *Il est aisé d'affecter un raccourci clavier à une macro.*

Personnaliser le ruban et la barre d'outils Accès rapide

Vous pouvez personnaliser tous les onglets du ruban ainsi que la barre d'outils Accès rapide. Pour ce faire, procédez comme suit :

1. Affichez le fichier PERSONAL.XLSB et cliquez sur l'onglet Fichier, puis sur Options. Dans la boîte de dialogue, choisissez Personnaliser le ruban ou Barre d'outils Accès rapide.
2. Dans la liste Choisir les commandes dans les catégories suivantes, choisissez Macros (figure 11-2).

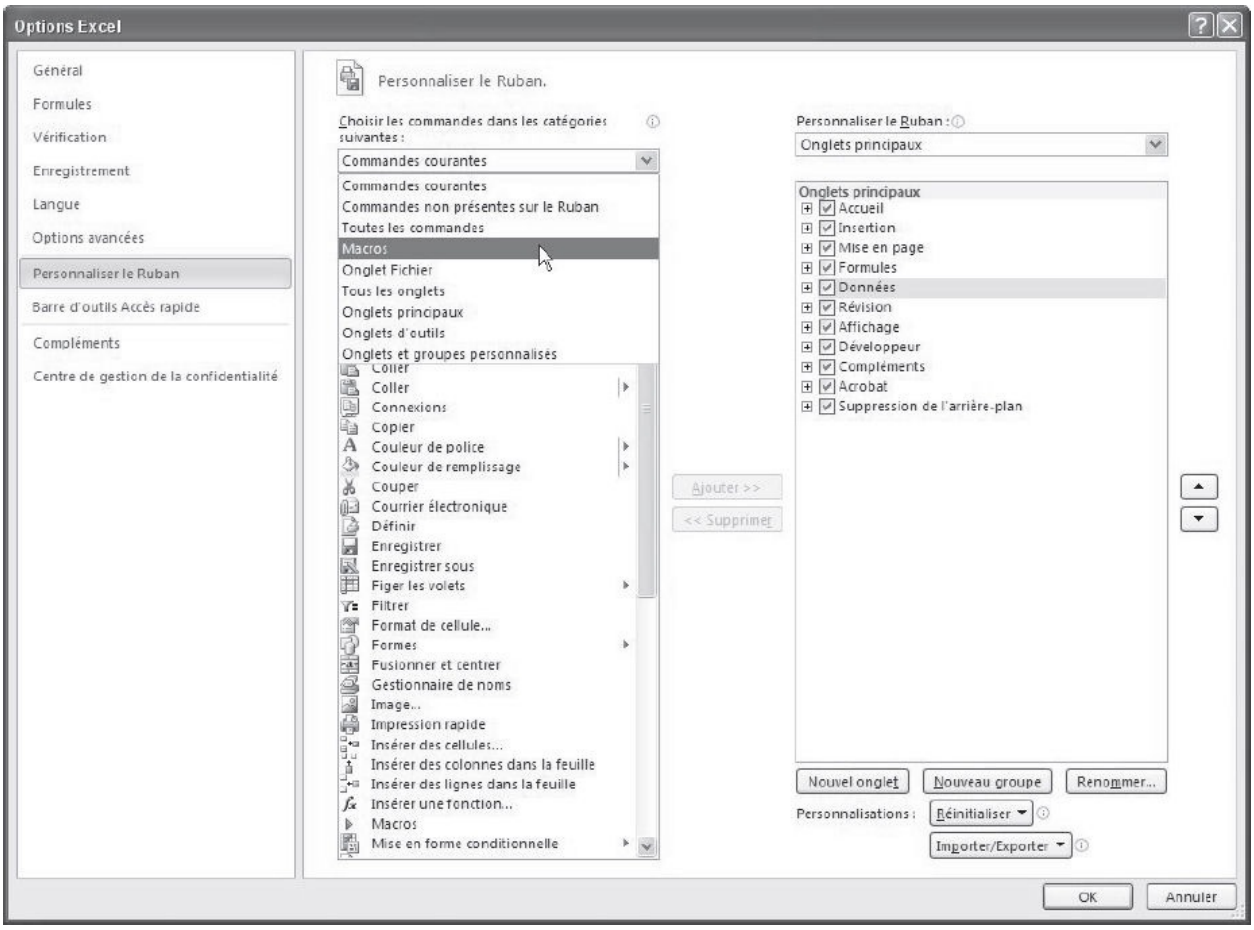


Figure 11-2 – *Depuis la version 2010 d'Excel, il est possible de personnaliser la barre d'outils Accès rapide, mais aussi le ruban.*

3. Dans la zone Personnaliser le ruban, sélectionnez l'une des trois options disponibles : Tous les onglets, Onglets principaux ou Onglets d'outils.
4. Choisissez l'élément que vous voulez personnaliser :
   – onglet existant : sélectionnez l'onglet dans la liste, puis Nouveau groupe ;
   – nouvel onglet : cliquez sur le bouton Nouvel onglet.
5. Cliquez-droit sur le nouvel onglet créé et choisissez Renommer. Dans la boîte de dialogue qui s'affiche, saisissez un nom approprié (voir figure 11-3), puis validez. Procédez de même pour le libellé Nouveau groupe (Personnalisé), situé sous l'onglet que vous venez de créer.

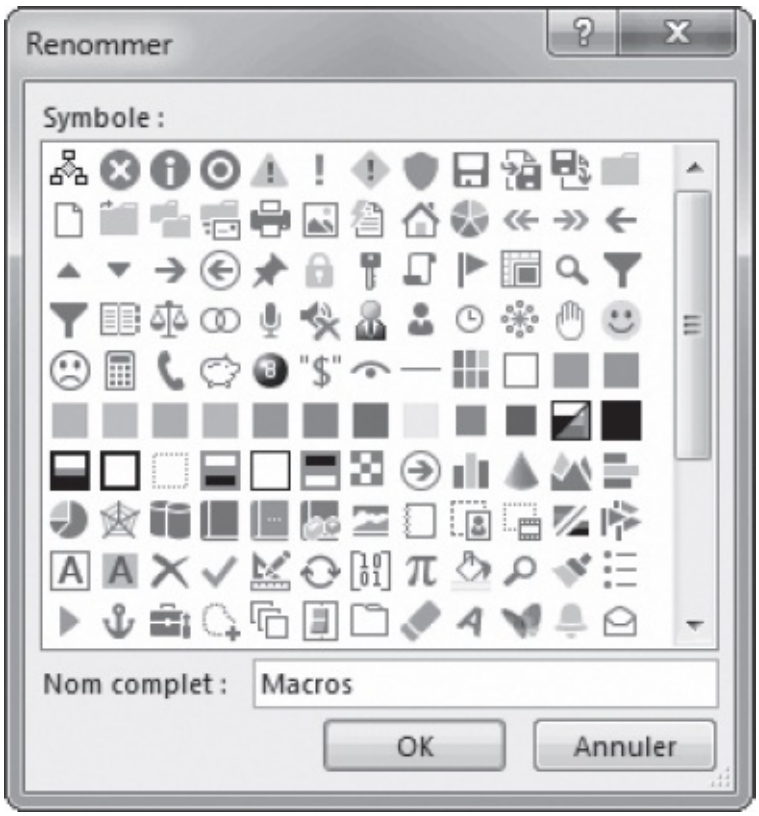


Figure 11-3 – *Si vous choisissez d'affecter vos macros à un nouvel onglet, donnez-lui un nom représentatif.*

Astuce

Pour modifier la position d'un onglet sur le ruban, sélectionnez-le et déplacez-le avec les flèches haut et bas.

6. Sélectionnez la macro de votre choix dans le volet gauche, puis cliquez sur Ajouter. Son nom apparaît dans le groupe sélectionné du nouvel onglet. Renommez la macro comme vous l'avez fait pour l'onglet et le groupe à l'étape 5, puis validez.
7. Enfin, utilisez les flèches situées sur la droite de la fenêtre pour définir la position de votre commande sur la barre d'outils, puis validez.

Info

Pour supprimer une commande personnalisée, cliquez-droit dessus et choisissez la commande Supprimer dans le menu contextuel.

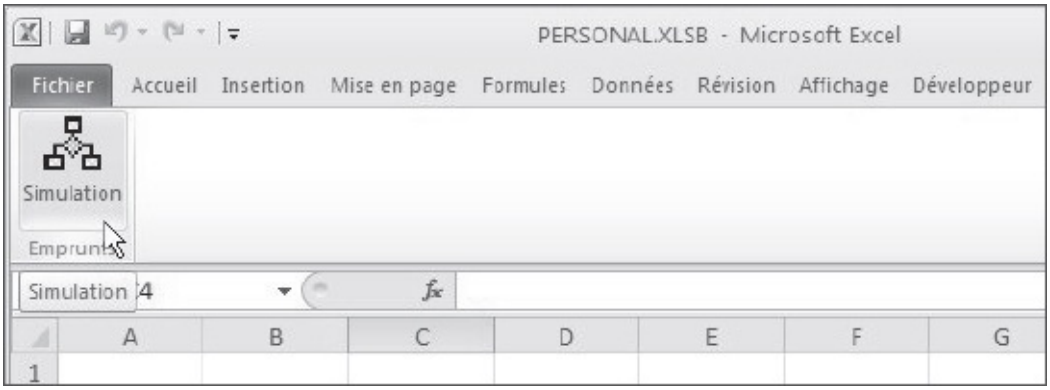


Figure 11-4 – *La macro est maintenant accessible via le nouvel onglet.*

Affecter une macro à un bouton

Il peut être pertinent d'affecter une macro à un bouton que vous aurez dessiné sur une feuille de calcul. En attribuant à ce bouton un libellé évocateur, vous invitez l'utilisateur à cliquer dessus sans qu'il sache que cela déclenchera l'exécution de la macro.

Commencez par dessiner le bouton sur la feuille de calcul Excel :

1. Affichez l'onglet Développeur.
2. Dans le groupe Contrôles, cliquez sur le bouton Insérer des contrôles et sélectionnez Bouton de commande parmi les contrôles affichés.
3. Redimensionnez le bouton et placez-le à l'endroit approprié sur la feuille de calcul.
4. Double-cliquez sur le bouton. Visual Basic s'ouvre sur le code suivant :

```
Private Sub CommandButton1_Click()
End Sub
```

   Il s'agit d'une procédure événementielle. Nous étudierons ce type de procédure en détail dans le chapitre suivant, lors de la création d'interface utilisateur. Pour l'instant, contentez-vous d'y placer le code que vous souhaitez exécuter lorsque l'utilisateur cliquera sur le bouton de commande.

   Vous pouvez rédiger directement le code ou appeler la procédure de votre choix en plaçant simplement une instruction `Call`.

5. Affichez la fenêtre Propriétés et affectez le libellé du bouton à l'attribut `Caption`.
6. Revenez à Excel et quittez le mode Création en cliquant sur le bouton du même nom du groupe Contrôles (onglet Développeur).

Affecter une macro à un objet

Une macro peut également être affectée à un objet tel qu'un graphique Excel ou un dessin de la couche Dessin des applications Office. Pour cela, cliquez-droit sur l'objet voulu et choisissez la commande Affecter une macro du menu contextuel.

Le curseur se transforme alors en main lorsqu'il est placé au-dessus de l'objet en question. Un clic sur celui-ci déclenche la macro.

Info

Pour manipuler les commandes de menu et les barres d'outils Excel par programmation – par exemple, ajouter ou supprimer des commandes selon le contexte –, utilisez la propriété `CommandBars`. Pour plus de précisions, consultez l'aide en ligne.

TROISIÈME PARTIE

Développer des interfaces utilisateur

12

Créer des interfaces utilisateur

Les feuilles constituent un élément essentiel de la programmation Visual Basic. Ce sont des zones sur lesquelles vous placez des contrôles tels que des cases à cocher, des boutons d'option, des zones de texte, des boutons de commande, etc. Ils forment une interface graphique simple et intuitive entre l'utilisateur et le programme.

Un événement utilisateur, tel qu'un clic de souris ou une modification de valeur, qui touche un contrôle est automatiquement repéré par le programme et lance l'exécution du code que vous lui avez associé. On parle alors de *procédure événementielle* ou *procédure d'événement*.

Les contrôles prennent une valeur, déterminée par leur état (case cochée ou non cochée, texte d'une zone de texte, etc.). L'exploitation des feuilles consiste généralement à passer à une procédure de code les informations que l'utilisateur entre sur la feuille (les valeurs des différents contrôles) et valide – en cliquant sur un bouton libellé OK, par exemple. Ces données sont ensuite exploitées par le programme.

Les phases de développement de feuilles

Les feuilles ramènent des tâches complexes à la simple information des champs d'une boîte de dialogue pour l'utilisateur final.

La création de feuilles VBA se déroule en trois phases :

- Détermination des besoins. Avant de vous lancer dans la création d'une feuille, réfléchissez aux fonctions que devra jouer l'interface développée. Quels doivent en être les différents contrôles ? Comment seront-ils organisés sur la feuille ? Quel seront les types d'interaction entre les contrôles ? À quels événements utilisateur devront-ils répondre ? Nous vous conseillons de réaliser sur papier un dessin approximatif de la feuille et de noter les informations essentielles sur les différents contrôles la composant

avant d'entamer la deuxième phase.

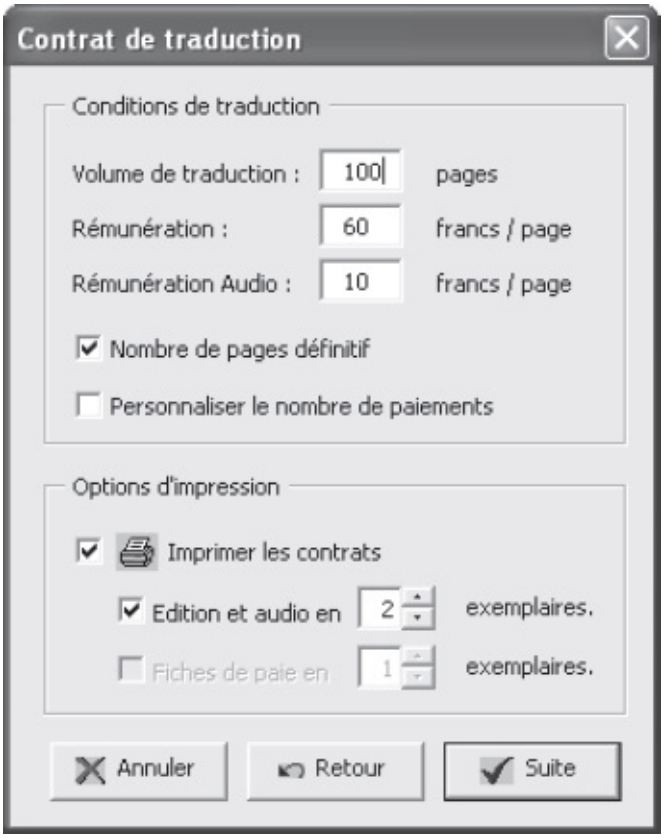


Figure 12-1 – *Les feuilles constituent l'interface graphique de vos projets VBA.*

- Création visuelle de la feuille. Durant cette phase, vous allez dessiner votre feuille dans Visual Basic Editor. Vous créerez la feuille et y placerez les contrôles voulus (cases à cocher, boutons d'option). Vous paramétrerez les propriétés de la feuille et des contrôles qui la composent, afin d'en déterminer l'apparence (couleur, taille, texte par défaut) et le comportement (accessibilité, nombre de caractères autorisés, etc.).
- Écriture du code attaché à la feuille. Sans code affecté aux différents éléments qui la composent, une feuille n'est qu'une jolie boîte de dialogue sans fonctionnalité. Durant cette phase, vous déterminerez son comportement face aux différents événements susceptibles de l'affecter et écrirez le code exploitant les données fournies par l'utilisateur.

Ce chapitre présente la création de feuilles. Vous y découvrirez les différents contrôles à votre disposition, apprendrez à les placer et à utiliser les outils de VBA pour créer des interfaces à l'apparence professionnelle. Les outils de développement visuel de feuilles de Visual Basic facilitent la création

d'interfaces complètes, semblables aux boîtes de dialogue des applications Office.

Créer une feuille

Le développement de feuilles se fait dans la fenêtre UserForm de Visual Basic Editor.

Pour créer une nouvelle feuille :

1. Ouvrez l'Explorateur de projets et sélectionnez celui dans lequel vous souhaitez créer une feuille.

2. Cliquez sur la flèche du bouton Ajouter de la barre d'outils Standard et sélectionnez UserForm ou sélectionnez la commande UserForm du menu Insertion ou cliquez-droit sur n'importe quel élément du projet dans l'Explorateur de projet et sélectionnez Insertion > UserForm.
3. Une fenêtre UserForm s'ouvre (voir figure 12-2). La feuille et la boîte à outils y sont affichées. Par défaut, la feuille est intitulée UserForm1, UserForm2 s'il existe déjà une feuille UserForm1, etc.

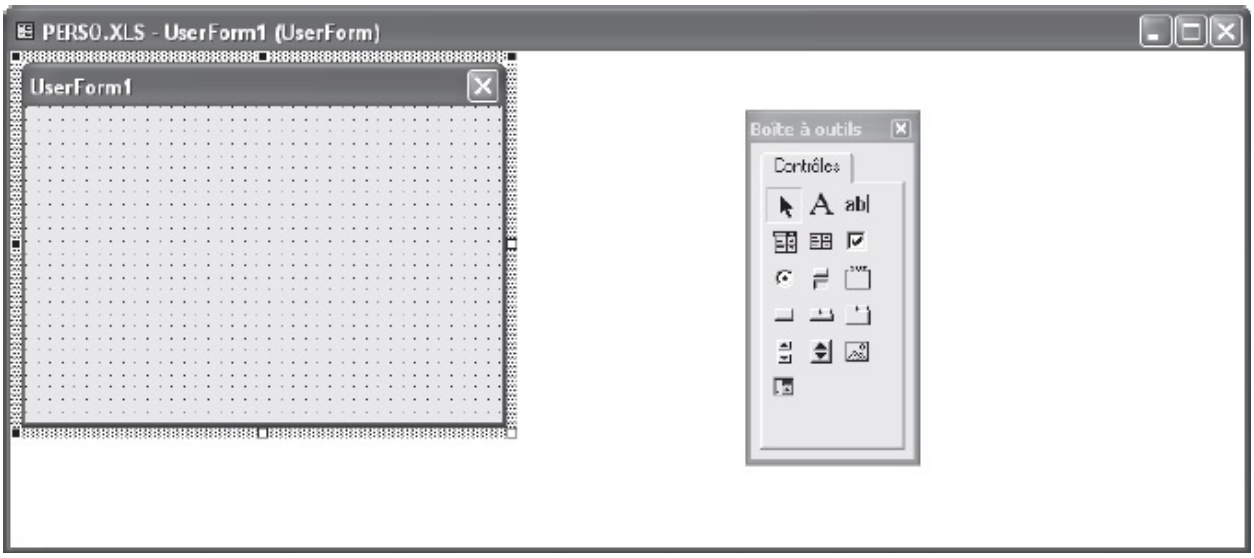


Figure 12-2 – *Votre feuille encore vierge de tout contrôle.*

La nouvelle feuille apparaît dans l'Explorateur de projet.

4. Ouvrez sa fenêtre Propriétés (F4) et renseignez ses propriétés `Name` (nom qui sera utilisé dans le code pour la référencer) et `Caption` (libellé de la feuille

qui apparaît dans sa barre de titre). La figure 12-3 présente une feuille dont la propriété `Caption` a été définie à « Exemple de feuille UserForm ».

5. Vous pouvez à tout moment utiliser les poignées de redimensionnement pour modifier la taille de la feuille.
6. Cliquez sur le bouton Enregistrer de la barre d'outils Standard.

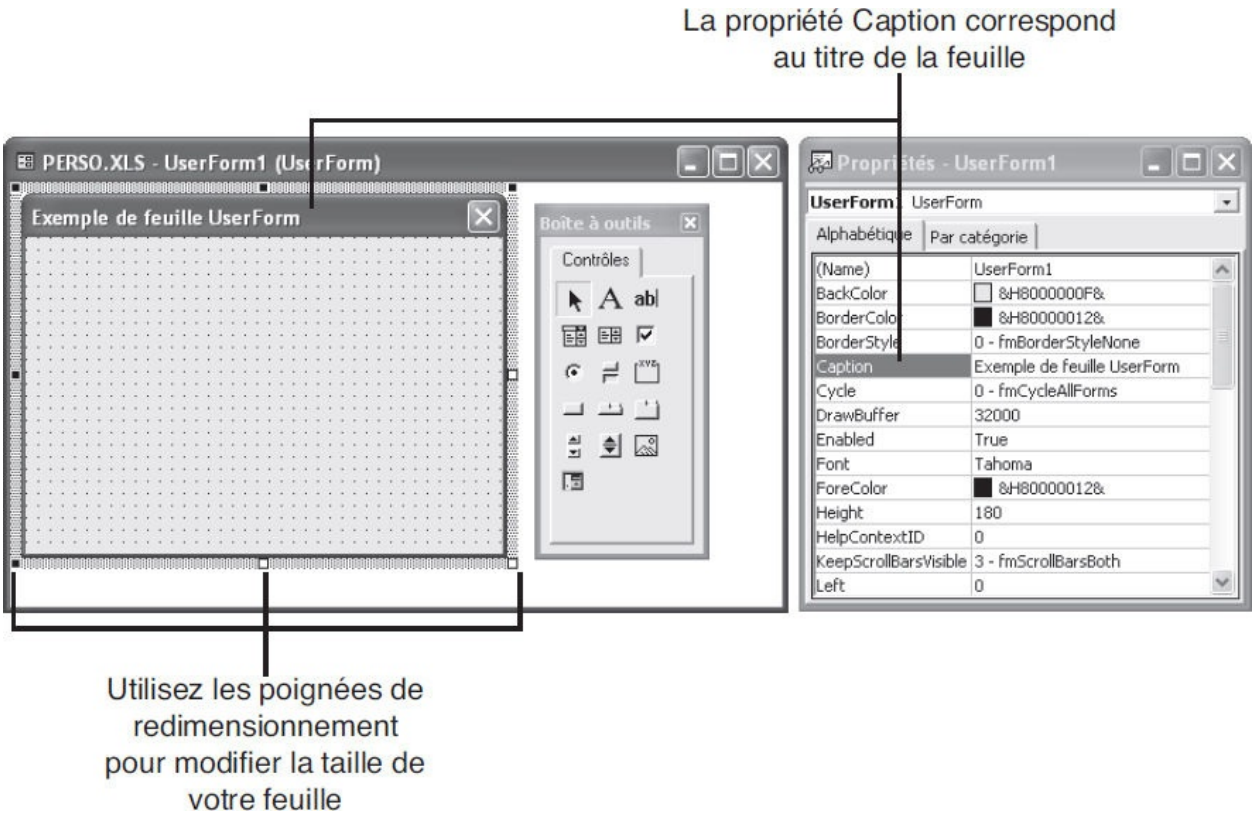


Figure 12-3 – *Définissez les propriétés Name et Caption de la feuille.*

Pour ouvrir la fenêtre UserForm d'une feuille existante, ouvrez l'Explorateur de projet (Ctrl+R) et double-cliquez sur celle-ci.

Les contrôles de la boîte à outils

La boîte à outils contient les contrôles que vous pouvez placer sur votre feuille. Au même titre que la feuille elle-même, les contrôles sont des objets. Vous n'avez pas à vous soucier de la façon dont ils fonctionnent. Il vous suffit d'en connaître les méthodes, propriétés et événements membres pour les exploiter.

Cliquez sur le bouton Boîte à outils de la barre Standard ou sélectionnez la commande Boîte à outils du menu Affichage.

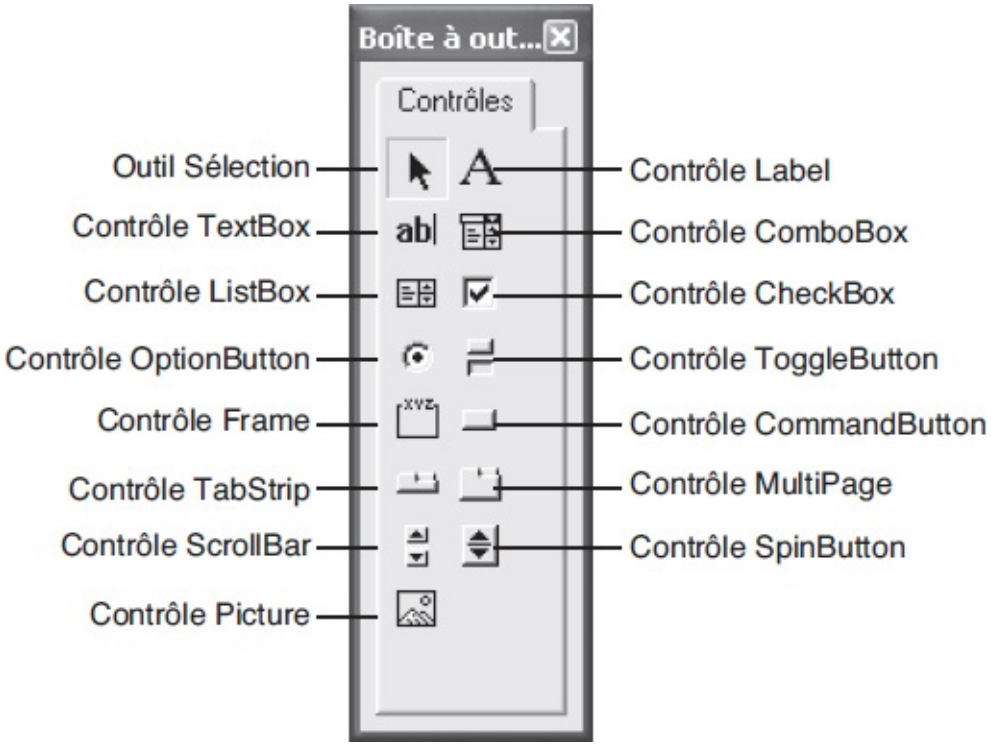


Figure 12-4 – *La boîte à outils contient des contrôles Windows usuels.*

Outil Sélection

L'outil Sélection sert à choisir un ou plusieurs contrôles sur une feuille. Il est activé par défaut dans la boîte à outils. Cet outil redevient actif dès que le contrôle choisi est déposé sur la feuille. Si vous sélectionnez un contrôle, puis décidez de ne plus le placer sur la feuille, cliquez sur l'outil Sélection.

Contrôle Label

Le contrôle `Label` place un intitulé sur la feuille, généralement à côté d'un contrôle ne possédant pas cet attribut (une zone de texte par exemple), pour aider l'utilisateur à en identifier la fonction.

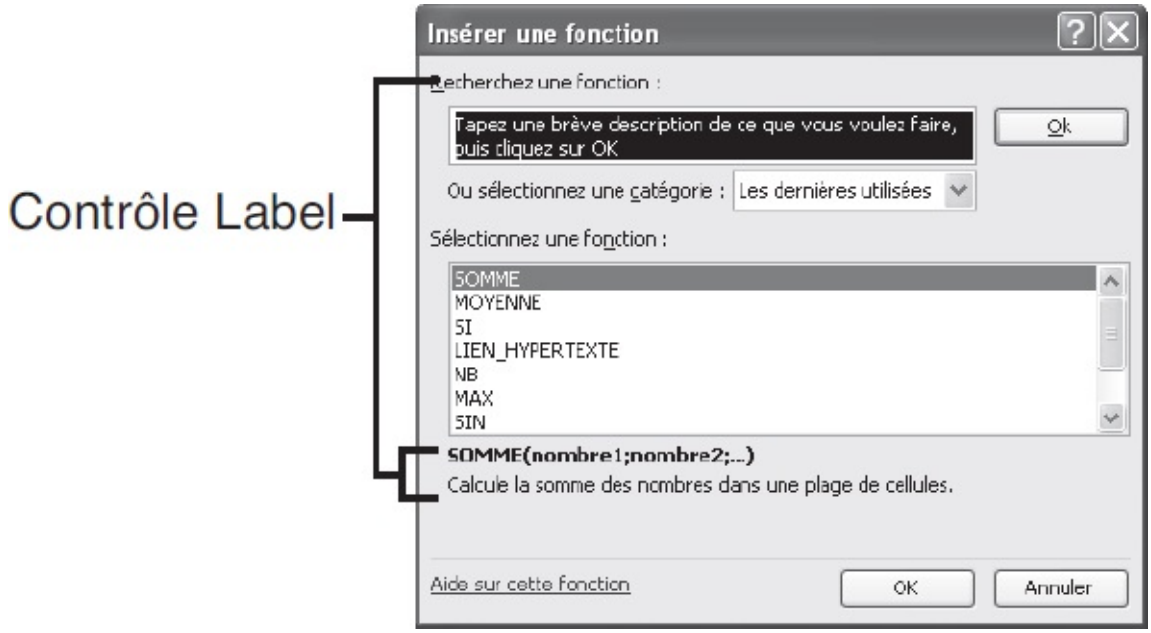


Figure 12-5 – *Le contrôle Label attache un libellé à un contrôle n'en possédant pas.*

Contrôle TextBox

Le contrôle `TextBox` place une zone de texte sur la feuille, dans laquelle l'utilisateur pourra saisir des informations. Les propriétés d'un `TextBox` permettent de contrôler le nombre maximal de caractères que pourra entrer l'utilisateur, l'affichage sur plusieurs lignes du texte ou encore le type d'alignement du texte. Elles sont étudiées au chapitre suivant.

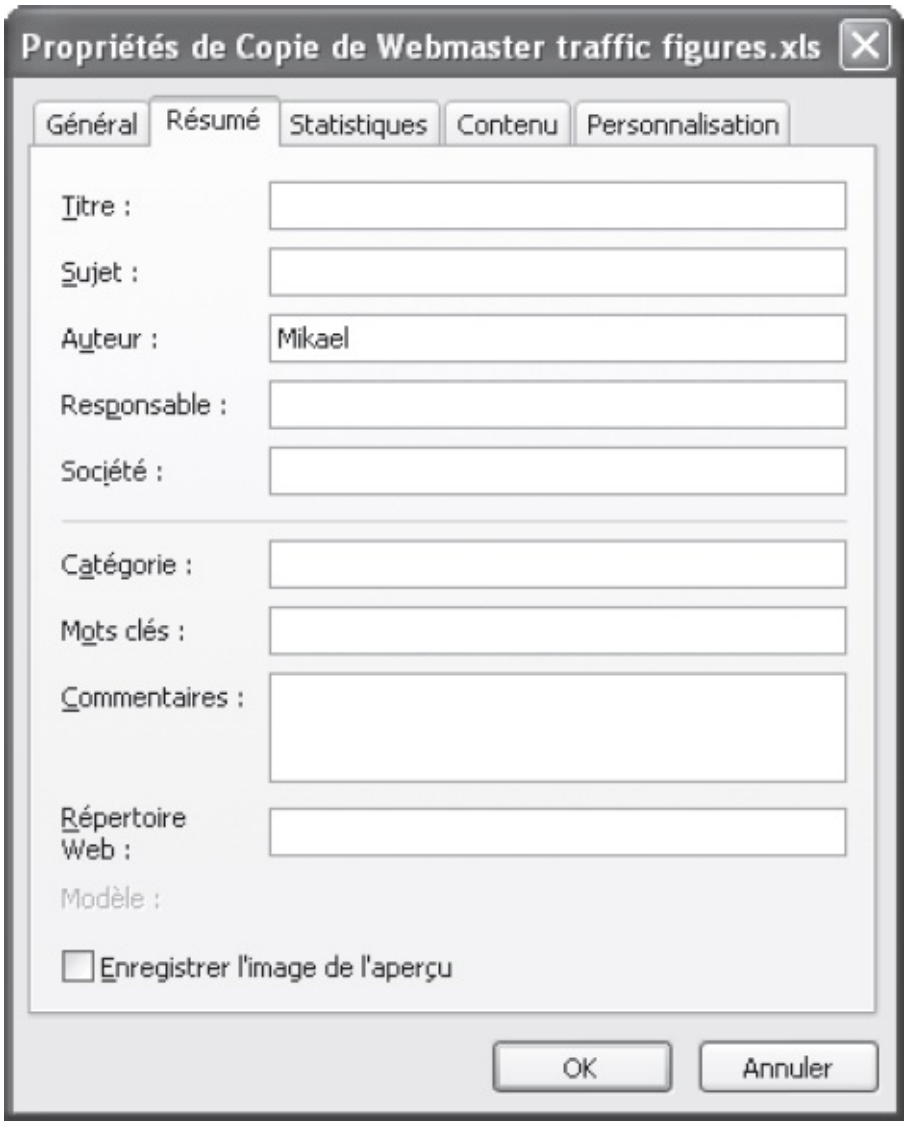


Figure 12-6 – *Le contrôle TextBox est l'un des contrôles les plus utilisés.*

Contrôle ComboBox

Le contrôle `ComboBox`, ou liste modifiable, insère une zone de texte permettant à l'utilisateur de saisir une valeur manuellement ou de la sélectionner dans la liste qui se déroule lorsqu'il clique sur le bouton prévu à cet effet. Un contrôle `ComboBox` peut autoriser ou non l'utilisateur à saisir une valeur ne figurant pas dans la liste.

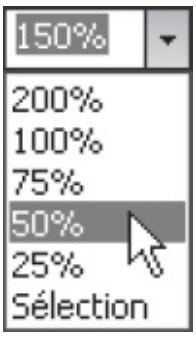


Figure 12-7 – *Le contrôle ComboBox permet à l'utilisateur de sélectionner une valeur dans la liste ou de choisir une valeur de son choix.*

Contrôle Frame

Le contrôle `Frame` place un cadre doté d'un intitulé sur une feuille, afin d'y placer des contrôles. Il est généralement utilisé pour distinguer aisément les catégories de contrôles d'une feuille. Les cadres servent aussi à associer des contrôles `OptionButton` (voir plus loin).

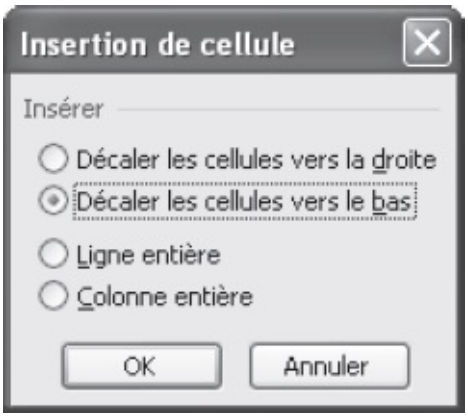


Figure 12-8 – *Un contrôle Frame groupe un ensemble de contrôles.*

Contrôle ListBox

Le contrôle `ListBox` insère une liste déroulante d'options parmi lesquelles l'utilisateur doit choisir. Les propriétés d'un `ListBox` permettent d'autoriser la sélection d'un seul ou de plusieurs éléments de la liste. Lorsque la hauteur n'est pas suffisante pour afficher tous les éléments, une barre de défilement verticale

lui est associée.

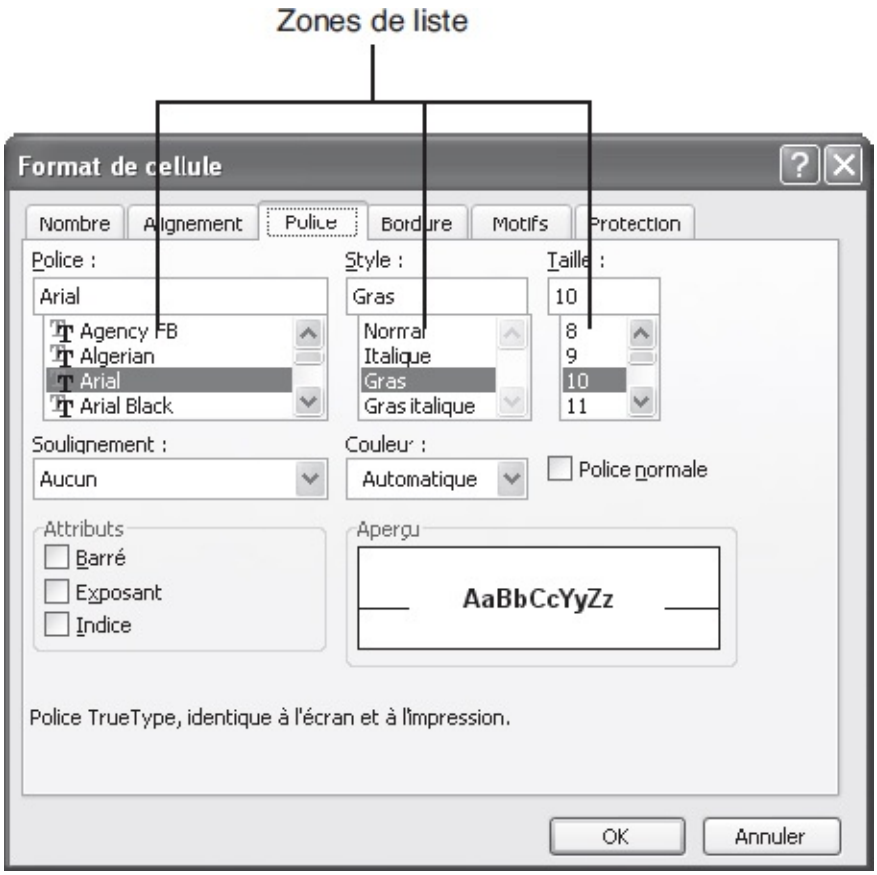


Figure 12-9 – *Une zone de texte peut être associée à une liste déroulante afin d'en afficher la valeur sélectionnée.*

Contrôle CheckBox

Un contrôle `CheckBox`, ou case à cocher, peut être activé (coché) ou désactivé (non coché) par l'utilisateur, afin de valider ou non une option. Les propriétés d'un `CheckBox` permettent aussi de lui faire accepter un troisième état, `Null` (ni coché, ni décoché) ; la case apparaît alors grisée.

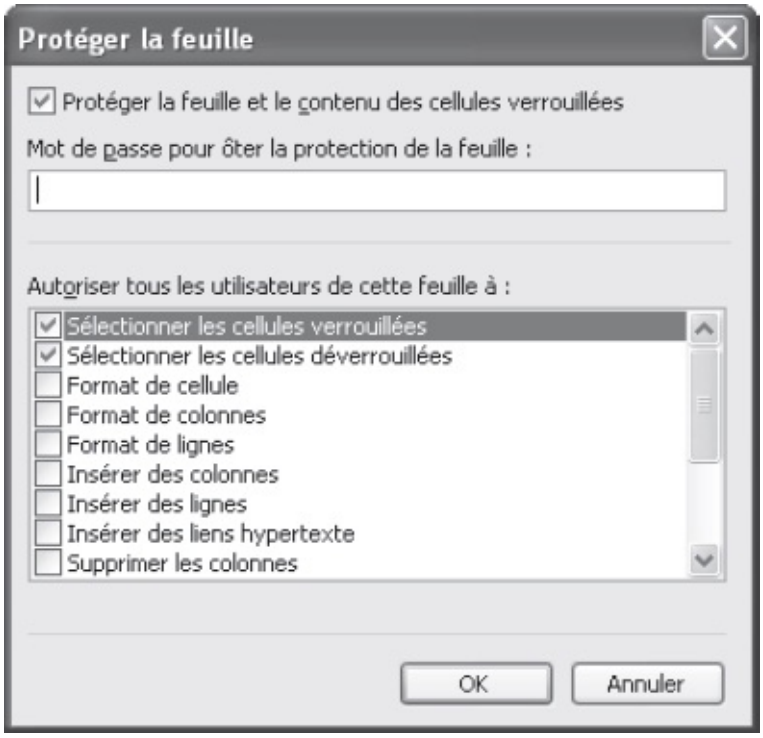

Protéger la feuille
Protéger la feuille et le contenu des cellules verrouillées
Mot de passe pour ôter la protection de la feuille :
Autoriser tous les utilisateurs de cette feuille à :
Sélectionner les cellules verrouillées
Sélectionner les cellules déverrouillées
Format de cellule
Format de colonnes
Format de lignes
Insérer des colonnes
Insérer des lignes
Insérer des liens hypertexte
Supprimer les colonnes
OK
Annuler

Figure 12-10 – *Les cases à cocher servent à valider ou non une option.*

Contrôle OptionButton

Les contrôles `OptionButton` proposent à l'utilisateur un choix parmi plusieurs options. Lorsque plusieurs contrôles `OptionButton` sont associés, seul l'un d'eux peut être activé.

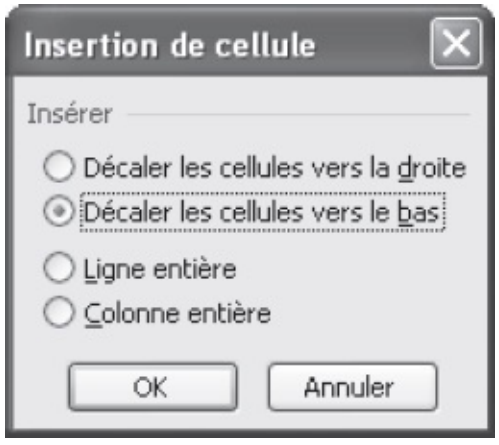


Figure 12-11 – *Un seul bouton d'option peut être sélectionné.*

Contrôle ToggleButton

Le contrôle `ToggleButton`, ou bouton bascule, sert à l'utilisateur pour activer ou désactiver une option, voire choisir l'état `Null` si les propriétés du contrôle l'y autorisent. Ce contrôle offre en fait les mêmes fonctionnalités qu'une case à cocher, avec l'apparence d'un bouton de commande. Lorsqu'un bouton bascule est activé, il semble enfoncé sur la feuille ; lorsqu'il est désactivé, il est saillant ; à l'état `Null`, il apparaît estompé. La figure 12-12 présente les trois états possibles pour un contrôle `ToggleButton`.

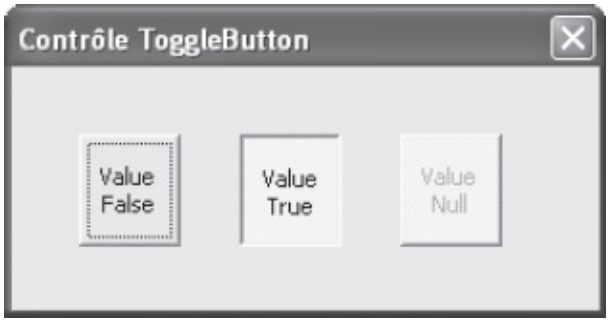


Figure 12-12 – *Le contrôle ToggleButton est assez rarement utilisé dans les interfaces de programme.*

Contrôle CommandButton

Le contrôle `CommandButton` est un bouton associé à une action. Il peut, par exemple, servir à valider les informations entrées dans la feuille (bouton OK) afin de passer à l'étape suivante du programme, ou au contraire interrompre le programme (bouton Annuler).

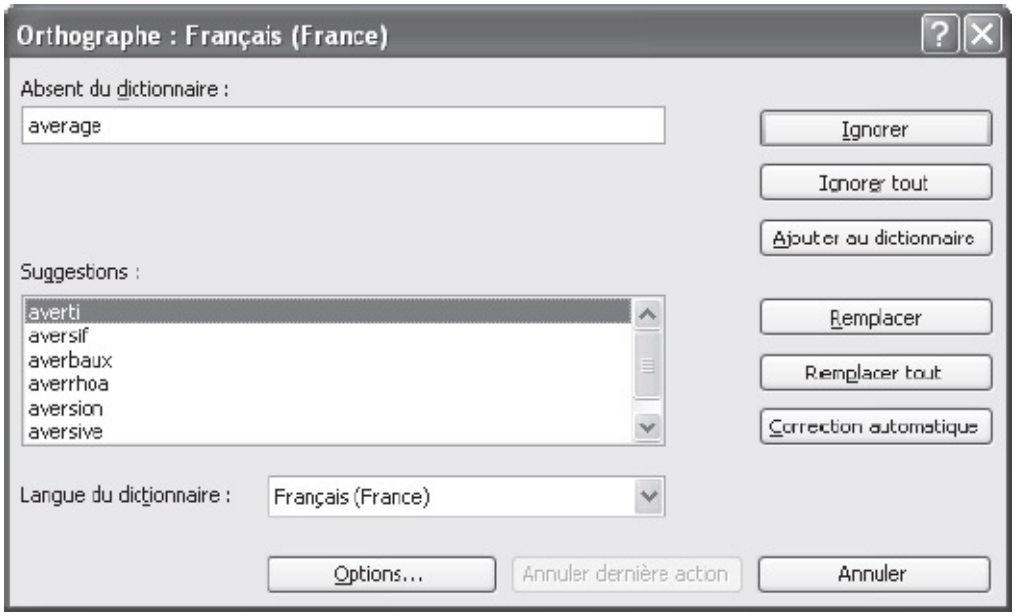


Figure 12-13 – *Les boutons de commande figurent parmi les contrôles les plus courants.*

Contrôle TabStrip

Le contrôle `TabStrip`, ou onglet, sert à créer une feuille dont l'affichage variera selon l'onglet sélectionné. Tout se passe comme si l'accès à plusieurs feuilles était dirigé à partir d'une seule feuille. Ce contrôle résout des problèmes d'espace en organisant les contrôles d'une feuille par types, classés sous différents onglets. Lorsque l'utilisateur clique sur un onglet, les contrôles de la feuille sont mis à jour par le code associé au contrôle.

Notez que les contrôles ne peuvent être associés à un onglet d'un `TabStrip`, ce dernier ne possédant pas un espace propre à chaque onglet. Les contrôles sont placés sur la feuille et leur mise à jour liée à la sélection d'un onglet ne peut être effectuée que par l'exécution du code précédemment écrit.

Contrôle MultiPage

Le contrôle `MultiPage` est semblable à `TabStrip`, en ce sens qu'il économise de l'espace en associant différents affichages à une même feuille. À la différence

de `TabStrip`, les pages constituant le contrôle `MultiPage` sont autonomes de la feuille. Autrement dit, lorsque vous placez des contrôles sur une page d'un contrôle `MultiPage`, ceux-ci ne s'afficheront que si la page en question est sélectionnée. Vous n'aurez donc pas à écrire de code pour faire varier l'affichage de la feuille, mais simplement à placer les contrôles voulus sur les pages constituant le contrôle.

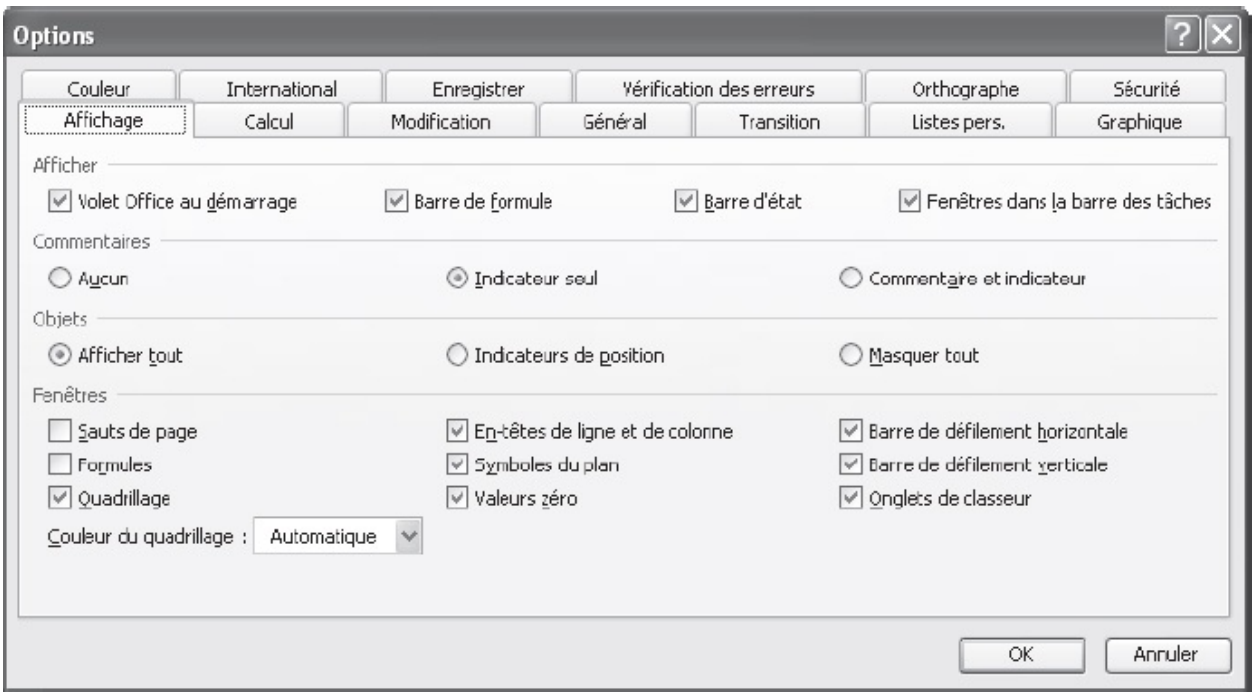


Figure 12-14 – *Plusieurs contrôles MultiPage peuvent être placés sur une feuille, chacun pouvant présenter un nombre variable d'onglets.*

Contrôle ScrollBar

Le contrôle `ScrollBar`, ou barre de défilement, offre à l'utilisateur une zone lui permettant de se déplacer dans un environnement. Une barre de défilement peut être horizontale ou verticale. Pour déterminer sa position, redimensionnez-la en utilisant ses poignées de sélection. On se déplace à l'aide d'une barre de défilement en cliquant sur l'une de ses touches fléchées ou à l'intérieur (entre le curseur et l'une des flèches), ou encore par un glisser-déplacer du curseur.

Un contrôle `ScrollBar` est automatiquement affecté à un `ListBox` lorsque la hauteur de ce dernier ne permet pas l'affichage de tous les éléments de la liste.

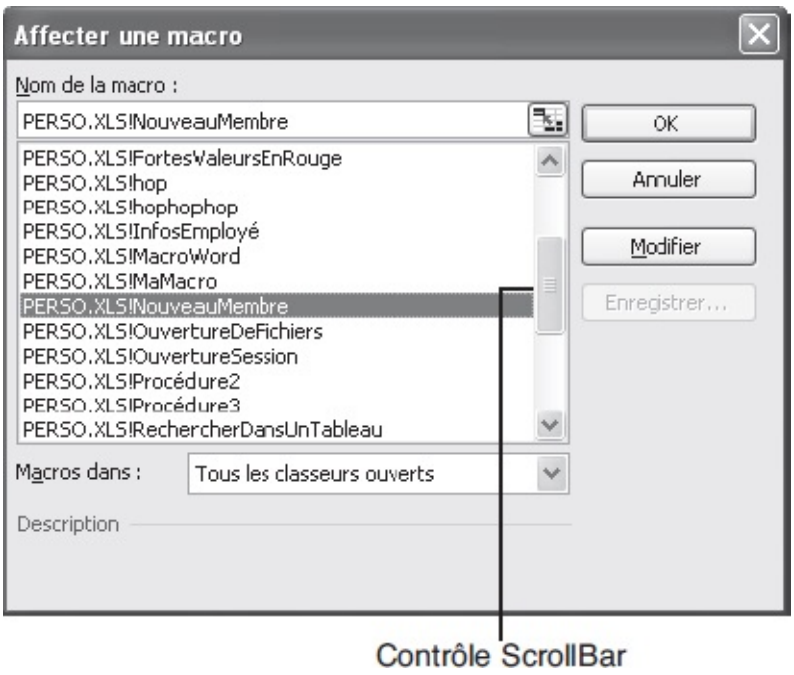


Figure 12-15 – *Un contrôle ScrollBar permet à l'utilisateur de se déplacer dans une zone de texte trop petite pour afficher la totalité de son contenu.*

Contrôle SpinButton

Un contrôle `SpinButton`, ou bouton toupie, est composé de deux flèches grâce auxquelles l'utilisateur sélectionne une valeur. Les boutons toupies sont généralement associés à une zone de texte dans laquelle s'affiche la valeur sélectionnée. Les propriétés de la zone de texte peuvent être définies pour que l'utilisateur puisse y entrer une valeur saisie au clavier ou seulement à l'aide du bouton toupie.

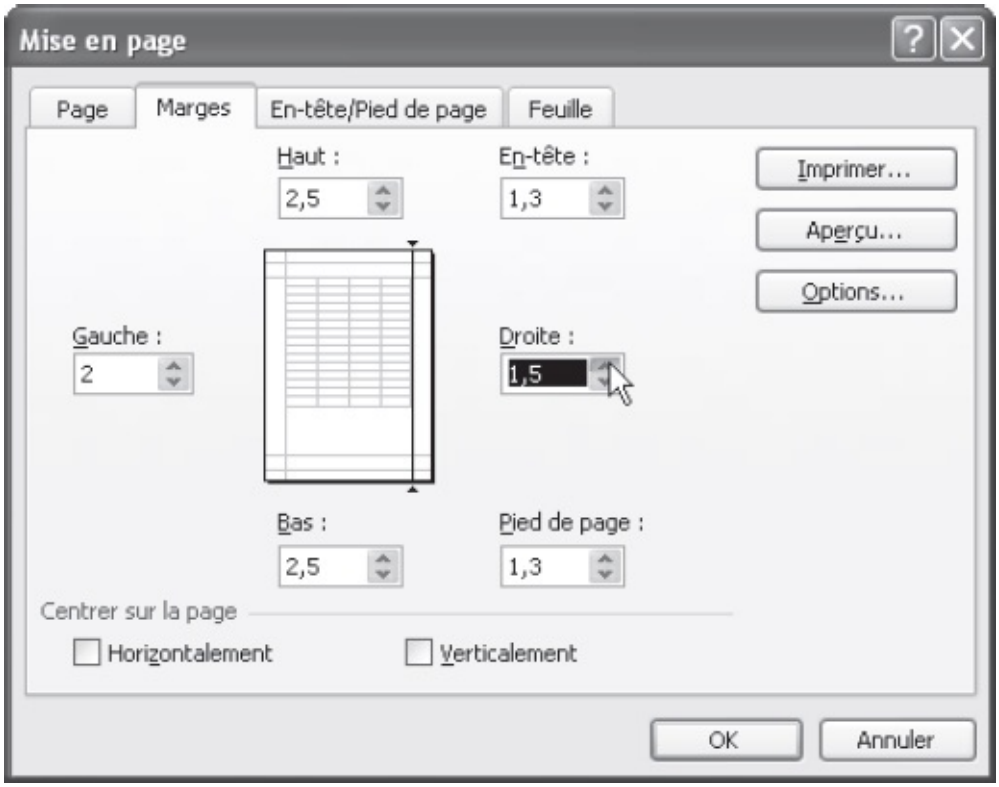


Figure 12-16 – *Un contrôle TextBox est généralement associé à un bouton toupie afin d'afficher la valeur choisie.*

Placer des contrôles sur une feuille

1. Dans la boîte à outils, cliquez sur le contrôle voulu ; il prend l'apparence d'un bouton enfoncé.
2. Déplacez le pointeur au-dessus de la feuille. Il se transforme en croix et une image représentant le type de contrôle sélectionné lui est accolée (voir figure 12-17).
3. Placez la croix à l'endroit où vous souhaitez insérer l'angle supérieur gauche du contrôle et cliquez. Le contrôle apparaît sur la feuille (voir figure 12-18) et l'outil Sélection est activé dans la boîte à outils.
4. Pour déplacer le contrôle, sélectionnez-le si nécessaire. Placez le curseur sur la bande grisée représentant la sélection, de façon qu'il prenne la même forme qu'à la figure 12-19, et glissez-déplacez-le jusqu'à l'emplacement voulu.
5. Pour redimensionner le contrôle, sélectionnez-le si nécessaire, puis tirez sur ses poignées de redimensionnement (voir figure 12-20).

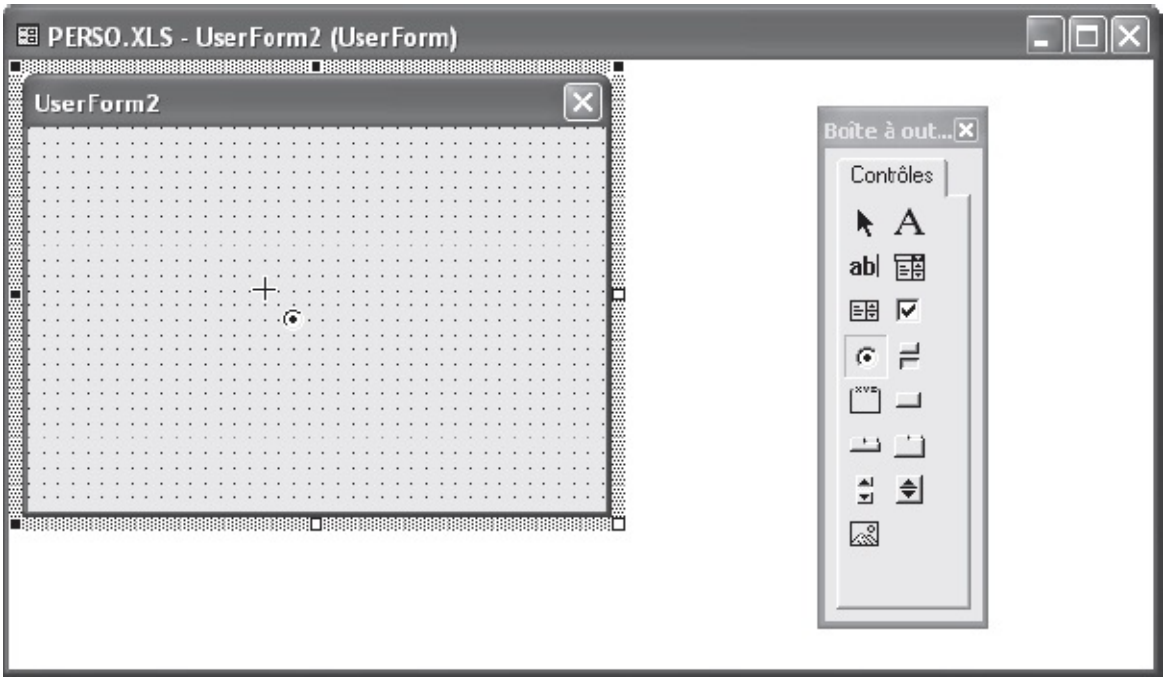


Figure 12-17 – *Une image accolée au pointeur indique le type de contrôle sélectionné.*

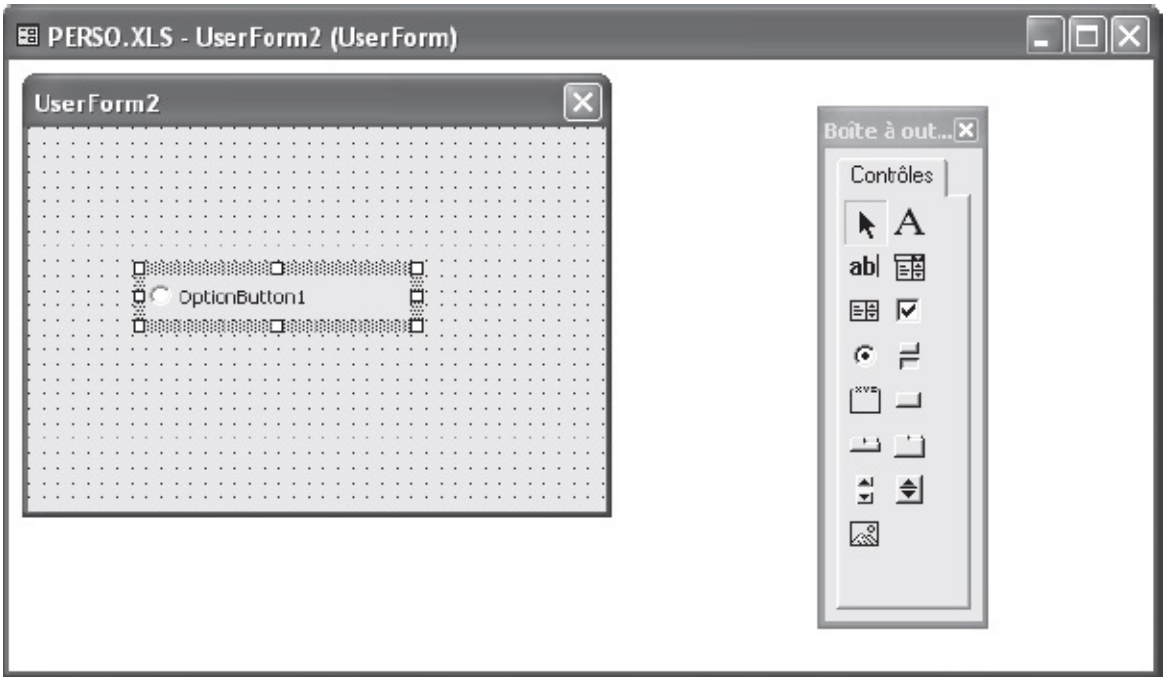


Figure 12-18 – *Le contrôle OptionButton déposé sur la feuille.*

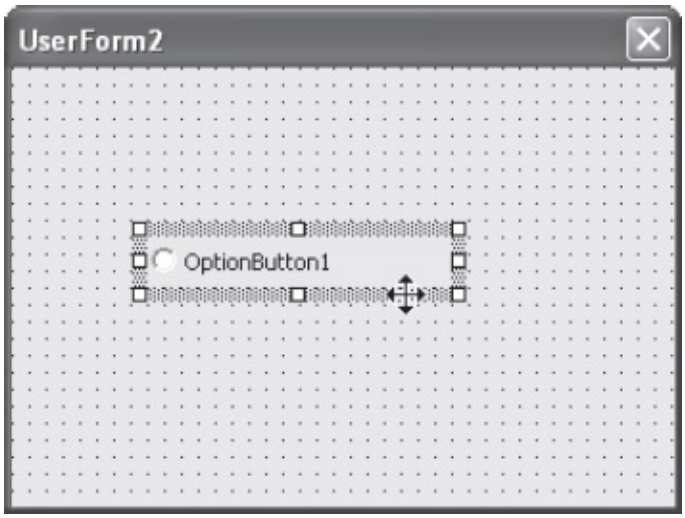


Figure 12-19 – *Cette forme de pointeur indique que vous allez déplacer le contrôle.*

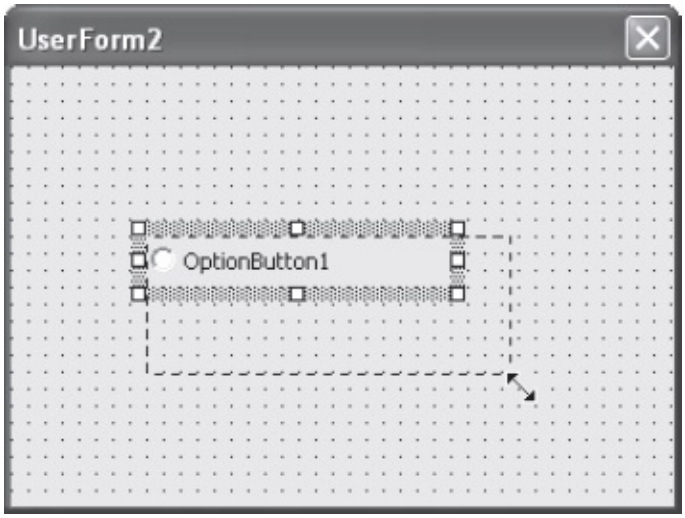


Figure 12-20 – *Utilisez les poignées de redimensionnement pour ajuster la taille du contrôle.*

Conseil

Dès que vous avez placé un contrôle sur une feuille, définissez sa propriété `Name`, qui correspond au nom utilisé dans le code pour faire référence à ce contrôle.

Astuce

Pour accéder à la rubrique d'aide Visual Basic d'un contrôle, sélectionnez ce dernier sur la feuille et tapez sur la touche F1.

Ce chapitre traite du développement de feuilles VBA. De la même façon, des

contrôles peuvent être placés sur une feuille de calcul Excel. Les procédures pour les exploiter sont les mêmes que pour une feuille UserForm développée dans Visual Basic Editor ; elles sont stockées dans l'objet Feuille portant le nom de la feuille, situé dans le module Objets Microsoft Excel.

Pour placer des contrôles sur une feuille de calcul plutôt que sur une feuille UserForm, affichez la barre d'outils Visual Basic dans Excel et cliquez sur le bouton Création, puis sur Boîte à outils Contrôles. Pour quitter ce mode, cliquez à nouveau sur le bouton Création.

Copier-coller des contrôles

Si vous souhaitez placer plusieurs fois le même contrôle sur une feuille, copiez-collez-le. Ainsi dupliqué, il héritera des propriétés du contrôle stocké dans le Presse-papiers, vous évitant de les redéfinir. N'oubliez pas de définir sa propriété `Name`.

Astuce

Pour copier-coller un contrôle, vous pouvez aussi cliquer-droit dessus et, tout en maintenant le bouton de la souris enfoncé, le faire glisser à l'endroit où vous souhaitez en placer une copie. Relâchez ensuite le bouton de la souris et, dans le menu contextuel qui s'affiche, sélectionnez Copier ici.

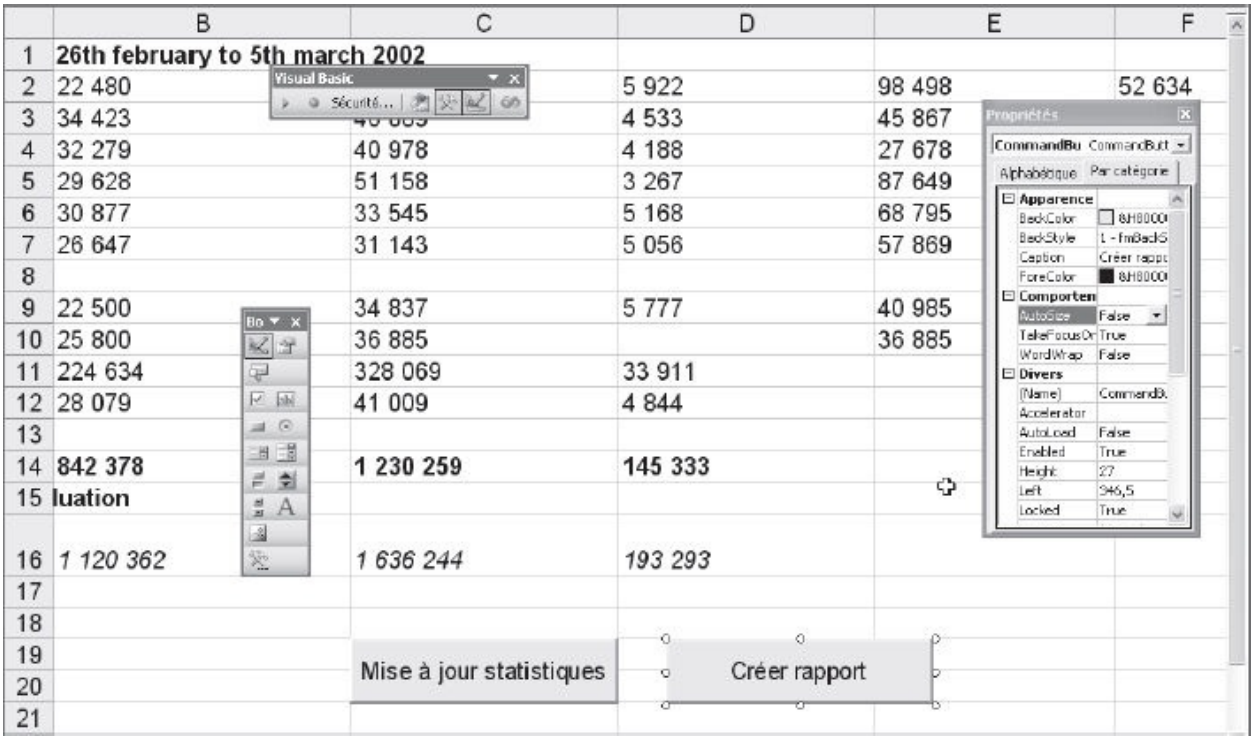


Figure 12-21 – *Vous pouvez aussi placer des contrôles sur une feuille de calcul Excel.*

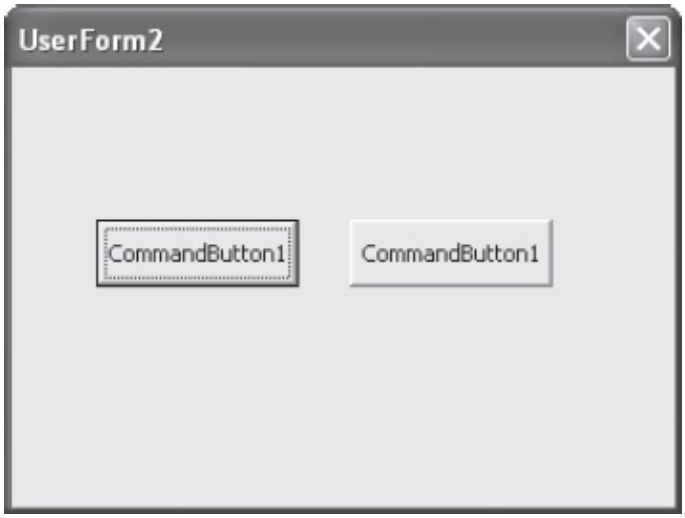


Figure 12-22 – *La copie du contrôle hérite des propriétés du contrôle source.*

Sélectionner plusieurs contrôles

- Pour sélectionner des contrôles contigus, cliquez sur le premier et, tout en appuyant sur la touche Maj, cliquez sur le dernier.
- Pour sélectionner des contrôles non contigus, cliquez sur le premier puis, tout en appuyant sur la touche Ctrl, cliquez successivement sur les autres.

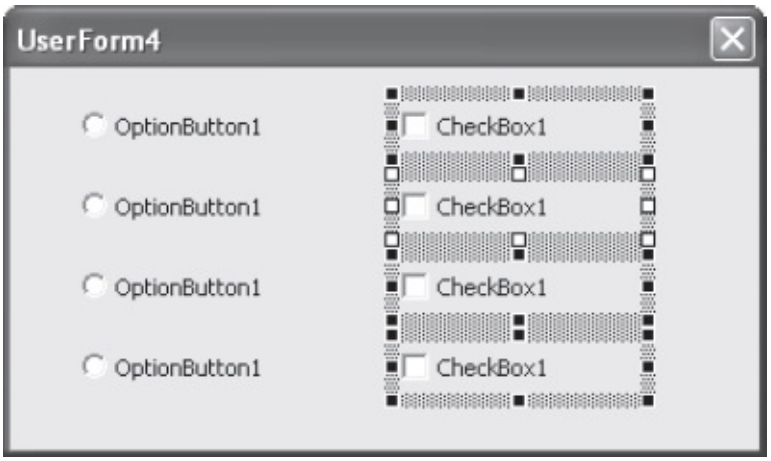


Figure 12-23 – *Pour sélectionner plusieurs contrôles sur une feuille, utilisez les touches Maj et Ctrl.*

Info

Lorsque vous sélectionnez plusieurs contrôles, les poignées de sélection de l'un d'eux apparaissent en blanc (celles des autres sont noires). Il s'agit du contrôle actif. Cette notion est particulièrement utile pour mettre en forme vos feuilles. Vous pouvez ainsi appliquer certaines propriétés du contrôle actif – telles que sa taille ou sa position – aux autres sélectionnés.

Supprimer des contrôles

Pour effacer des contrôles d'une feuille, sélectionnez-les et tapez sur la touche Suppr. Si vous souhaitez les placer dans le Presse-papiers avant de les coller dans une autre feuille, tapez Ctrl+X.

Mise en forme des contrôles

Le simple déplacement des contrôles sur une feuille à l'aide de la souris n'est souvent pas assez précis. Un certain nombre d'outils sont à votre disposition pour peaufiner la mise en forme : grille, commandes du menu Format ou de la barre d'outils UserForm (figure 12-24). Vous obtiendrez ainsi des contrôles parfaitement alignés et de dimensions proportionnées.

Figure 12-24 – *La barre d'outils UserForm.*

La grille

La grille désigne les points quadrillant une feuille dans une fenêtre UserForm, qui n'apparaissent qu'en phase de conception et ont pour fonction de faciliter la mise en place des contrôles.

Lorsque l'alignement sur la grille est activé, l'angle supérieur gauche des contrôles est automatiquement placé sur le point de la grille le plus proche. Les contrôles acceptent alors autant de positions qu'il existe de points de quadrillage sur la feuille.

1. Sélectionnez la commande Options du menu Outils de Visual Basic Editor et activez la page Général de la boîte de dialogue qui s'affiche. La zone Paramètres de grille de la feuille (voir figure 12-25) sert à définir les options.

2. Cochez les cases Afficher la grille afin de faire apparaître les points de quadrillage sur la feuille, puis déterminez la précision de la grille en spécifiant des valeurs de largeur et de hauteur.
3. Cochez la case Aligner les contrôles sur la grille, puis cliquez sur OK.

Info

La commande Ajuster à la grille du menu Format ajuste la hauteur et la largeur des contrôles sélectionnés aux lignes de grille les plus proches.

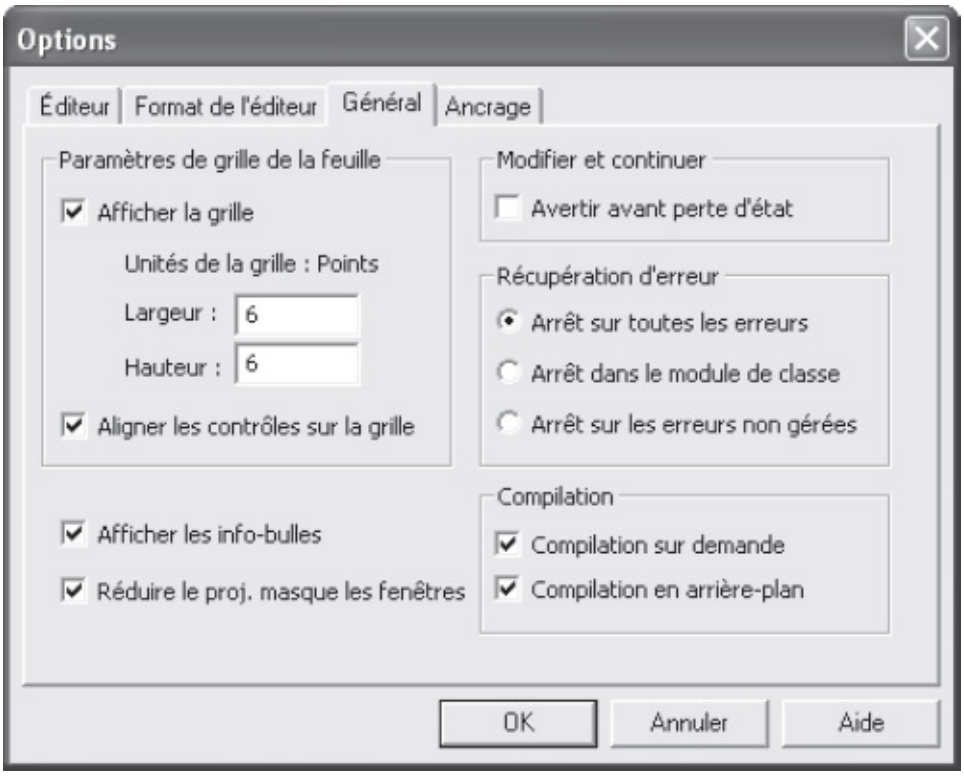


Figure 12-25 – *Paramétrez l'utilisation de la grille.*

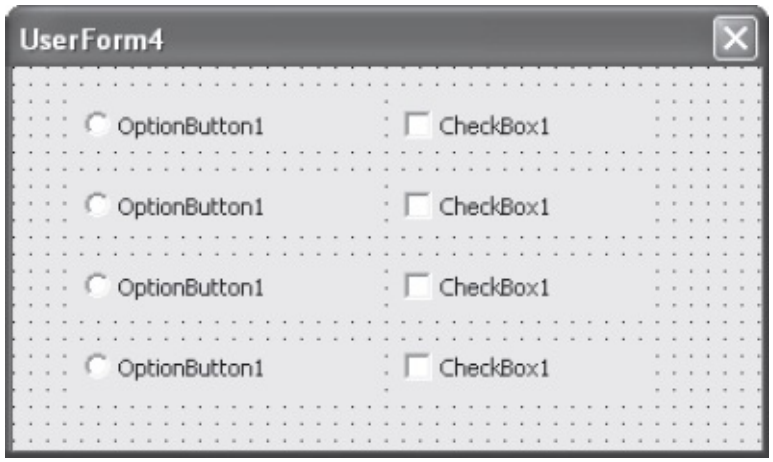


Figure 12-26 – *Quand ils sont fixés sur la grille les contrôles sont parfaitement alignés les uns par rapport aux autres.*

Aligner les contrôles

1. Sélectionnez les contrôles à aligner en prenant soin de repérer celui qui est actif (dont les poignées de sélection apparaissent en blanc).

> **Astuce**
> Pour modifier le contrôle actif d'une sélection, maintenez la touche Ctrl enfoncée et cliquez à deux reprises sur celui que vous souhaitez.

2. Choisissez la commande Aligner du menu Format ou cliquez sur la flèche du bouton Aligner de la barre d'outils UserForm (voir figure 12-27).
3. Choisissez l'alignement voulu parmi les sept proposés.

   Toute la sélection s'aligne sur le contrôle actif.

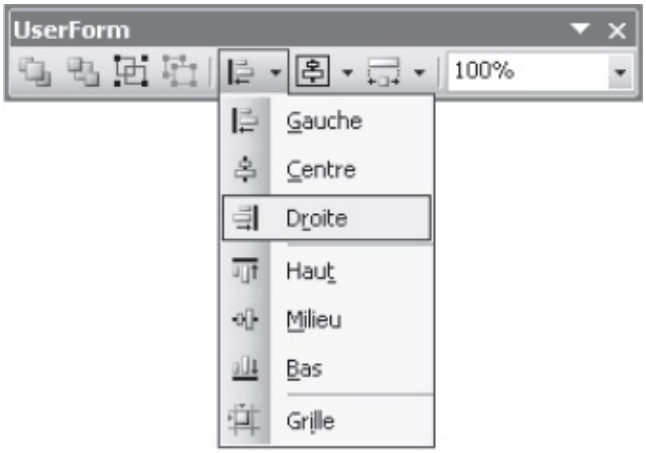


Figure 12-27 – *Sélectionnez un type d'alignement.*

> **Info**
> Un mauvais choix peut entraîner la juxtaposition des contrôles. Cliquez alors sur le bouton Annuler de la barre d'outils Standard ou tapez le raccourci Ctrl+Z pour ramener les contrôles à leur position précédente.

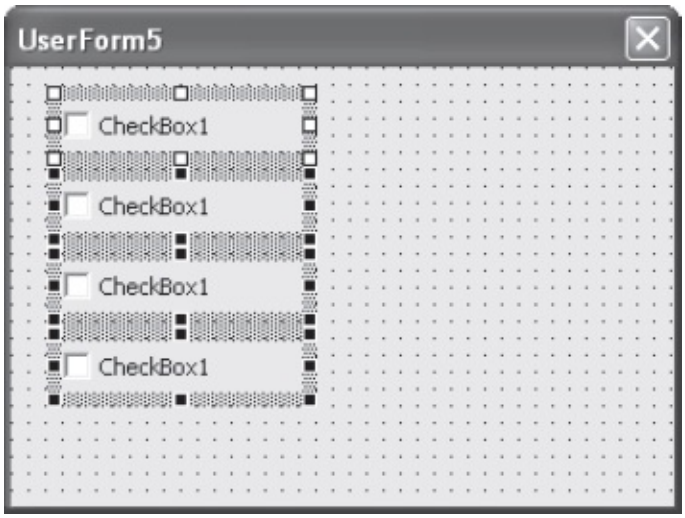


Figure 12-28 – *L'alignement des contrôles se fait sur le contrôle actif de la sélection.*

Uniformiser la taille des contrôles

1. Sélectionnez les contrôles.
2. Choisissez la commande Uniformiser la taille du menu Format ou cliquez sur la flèche du bouton Égaliser de la barre d'outils UserForm.
3. Choisissez d'uniformiser la hauteur des contrôles, leur largeur ou encore les deux.

Toute la sélection prend la taille du contrôle actif.

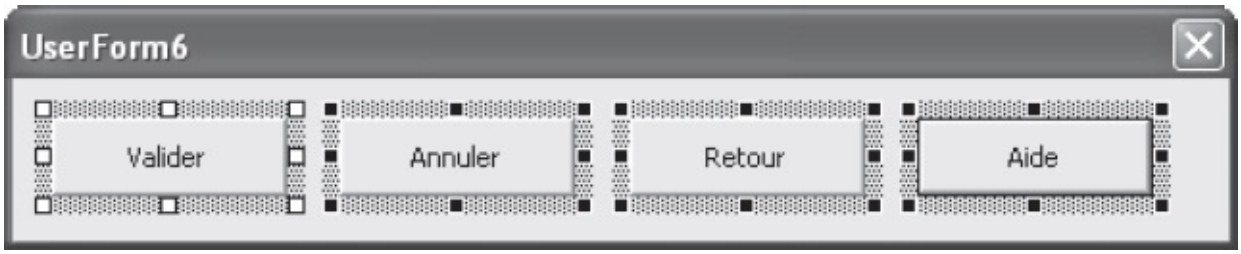


Figure 12-29 – *Une même largeur et une même hauteur ont été affectées aux contrôles sélectionnés.*

Astuce

Vous pouvez adapter la taille d'un contrôle à son contenu en choisissant la commande Ajuster la taille du menu Format.

Uniformiser l’espace entre les contrôles

1. Sélectionnez les contrôles.
2. Choisissez la commande Espacement horizontal ou Espacement vertical du menu Format.
3. Choisissez l’une des commandes du menu qui s’affiche :
   – Égaliser. L’espacement entre les contrôles sélectionnés sera le même.
   – Augmenter. L’espacement entre les contrôles sélectionnés augmentera d’une unité de grille, soit d’une distance égale à l’espace séparant deux points adjacents de la grille.
   – Diminuer. L’espacement entre les contrôles sélectionnés diminuera d’une unité de grille.
   – Supprimer. L’espace séparant les contrôles sélectionnés sera supprimé. Pour autant, les contrôles ne seront pas juxtaposés, mais accolés.

Astuce

Vous pouvez adapter la taille d’un contrôle à son contenu en choisissant la commande Ajuster la taille du menu Format.

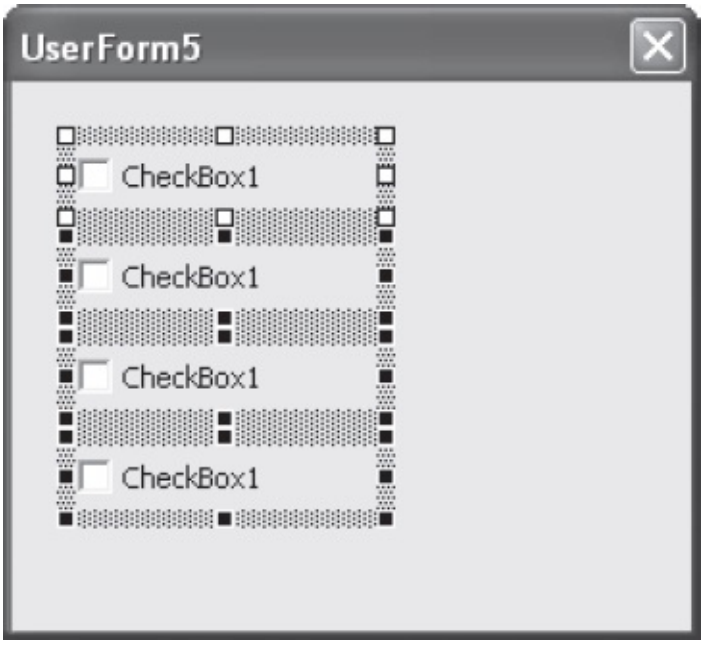


Figure 12-30 – *Un même espacement vertical a été appliqué aux contrôles sélectionnés.*

Centrer les contrôles

1. Sélectionnez les contrôles.
2. Choisissez la commande Centrer sur la feuille du menu Format ou cliquez sur la flèche du bouton Centrer de la barre d'outils UserForm.
3. Choisissez un centrage horizontal ou vertical.

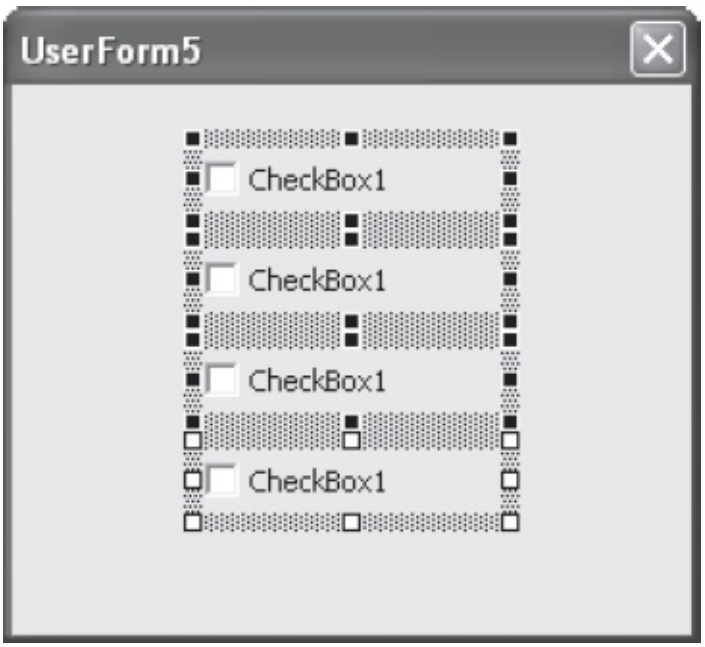


Figure 12-31 – *Les contrôles ont été centrés horizontalement sur la feuille.*

Réorganiser les boutons de commande

Pour placer les boutons dans la partie inférieure ou droite d'une feuille :

1. Sélectionnez les contrôles `CommandButton` à déplacer.
2. Choisissez la commande Réorganiser les boutons du menu Format.
3. Choisissez l'une des options de réorganisation :
   - En bas. Les boutons seront placés en bas et à gauche de la feuille et séparés par un même espace.
   - À droite. Les boutons seront placés à droite et en haut de la feuille et séparés par un même espace.

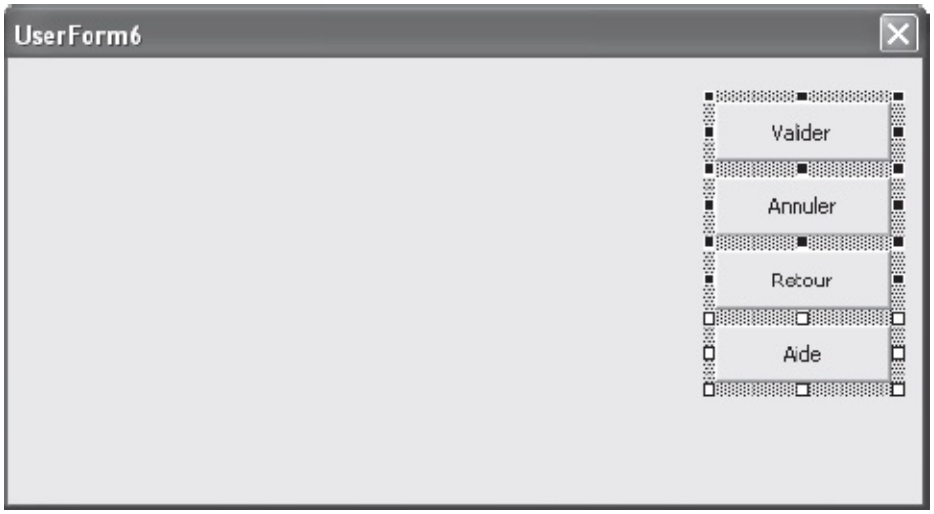


Figure 12-32 – *Les boutons ont été placés à droite de la feuille.*

Info

Si des contrôles autres que des boutons de commande sont sélectionnés, ils ne seront pas affectés par la commande Réorganiser les boutons. Si des contrôles occupent déjà l'espace vers lequel sont envoyés les boutons, ces derniers les recouvriront.

Grouper ou séparer des contrôles

Lorsque des contrôles sont groupés, un clic sur l'un d'entre eux sélectionne le groupe entier, un cadre les distinguant (voir figure 12-33). Un second clic sur l'un des contrôles du groupe le sélectionne. Vous pouvez ainsi agir sur ce dernier sans affecter le reste du groupe.

Grouper les contrôles présente plusieurs avantages :

- Les contrôles peuvent être sélectionnés en un seul clic et déplacés ou redimensionnés simultanément en agissant sur le cadre de sélection.
- La fenêtre Propriétés d'un groupe affiche les propriétés communes à tous les contrôles qu'il contient. La redéfinition de la valeur d'une propriété affecte alors tous les contrôles.

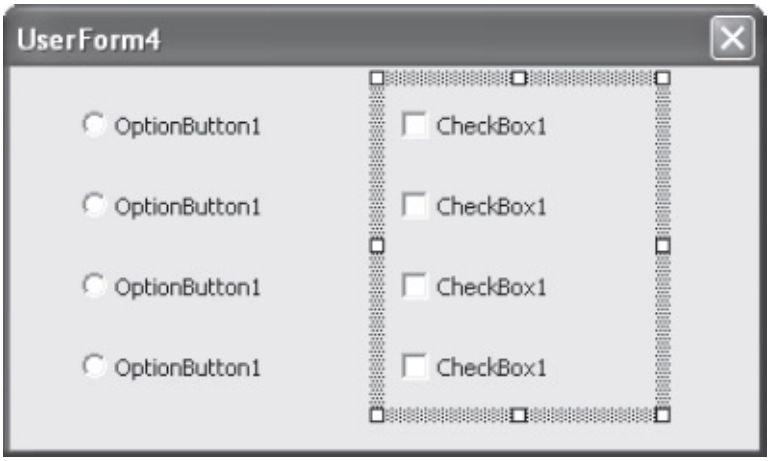


Figure 12-33 – *Un clic sur un contrôle sélectionne le groupe entier.*

Info

On ne peut sélectionner qu'un contrôle à la fois à l'intérieur d'un groupe. Déterminez la mise en forme des contrôles avant de les grouper.

Pour grouper des contrôles :

1. Sélectionnez les contrôles.
2. Choisissez la commande Grouper du menu Format ou cliquez sur le bouton Grouper de la barre d'outils UserForm.

Pour dissocier les contrôles d'un groupe :

1. Sélectionnez le groupe.
2. Choisissez la commande Séparer du menu Format ou cliquez sur le bouton Séparer de la barre d'outils UserForm.

Personnaliser la boîte à outils

Les contrôles les plus courants présentés précédemment sont disponibles par défaut dans la boîte à outils. Vous pouvez cependant personnaliser cette dernière en y ajoutant ou supprimant des contrôles.

Ajouter/supprimer un contrôle

Dans cet exemple, nous ajouterons le contrôle `Calendar` dans la boîte à outils. Comme il n'est plus livré avec les dernières versions d'Office, nous le fournissons avec les codes du livre, en téléchargement sur le site d'Eyrolles

(http://www.editions-eyrolles.com/dl/0067401). Si vous n'avez pas encore téléchargé les codes sources, commencez par le faire. Référencez ensuite le contrôle `Calendar`. Procédez comme suit :

1. Placez le fichier mscal.ocx (situé dans le dossier Bonus des codes sources du livre) dans le dossier de votre choix.
2. Activez l'onglet Développeur du ruban d'Excel.
3. Cliquez sur le bouton Insérer du groupe Code, puis sur le bouton Autres contrôles (voir figure 12-34)

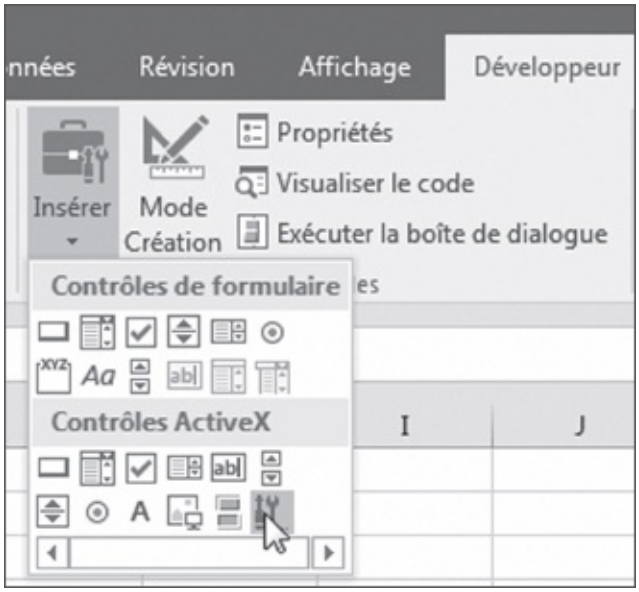


Figure 12-34 – *Choisissez la commande Autres contrôles.*

4. Dans la boîte de dialogue qui s'affiche, cliquez sur le bouton Enregistrer le contrôle personnalisé. Localisez le fichier mscal.ocx, puis validez.

   `Calendar` apparaît maintenant dans la boîte de dialogue Autres contrôles (voir figure 12-35).

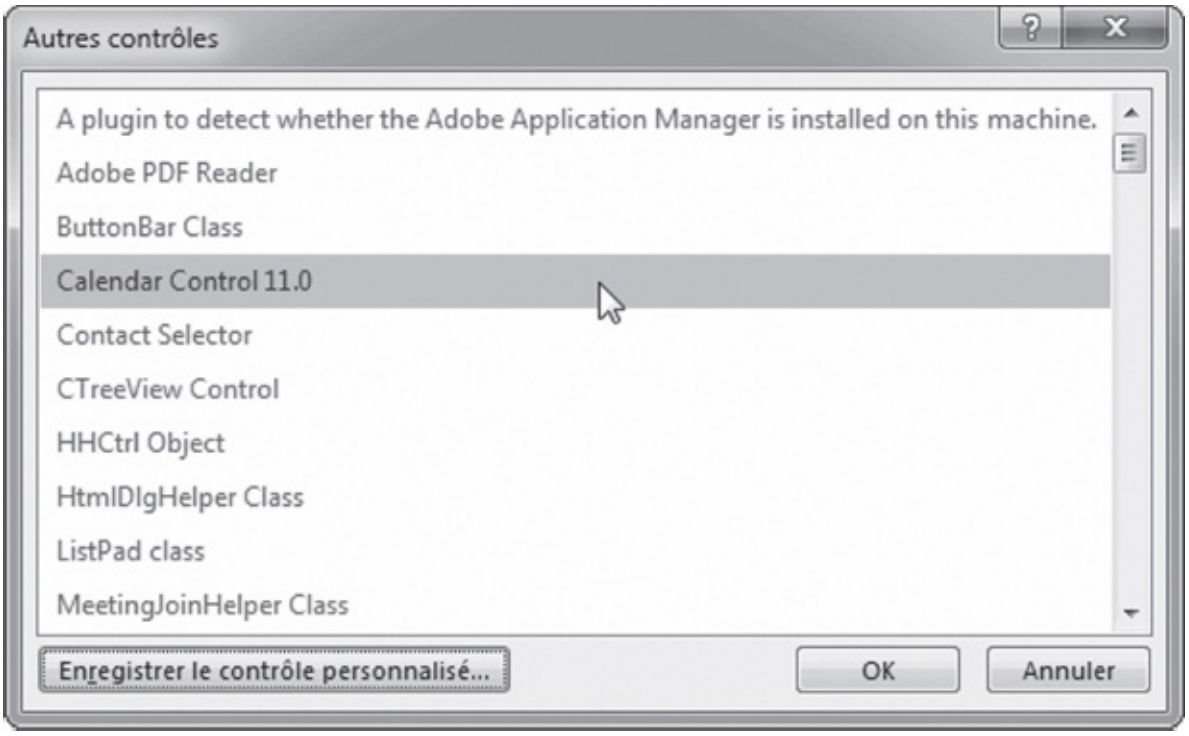


Figure 12-35 – *Le contrôle Calendar est maintenant référencé.*

Pour ajouter `Calendar` à la boîte à outils, ouvrez Visual Basic Editor et procédez comme suit :

1. Affichez la boîte à outils. Cliquez-droit sur celle-ci et sélectionnez la commande Contrôles supplémentaires du menu contextuel ou choisissez la commande Contrôles supplémentaires du menu Outils (voir figure 12-36). Les contrôles qui peuvent être exploités et placés sur la boîte à outils s'affichent.

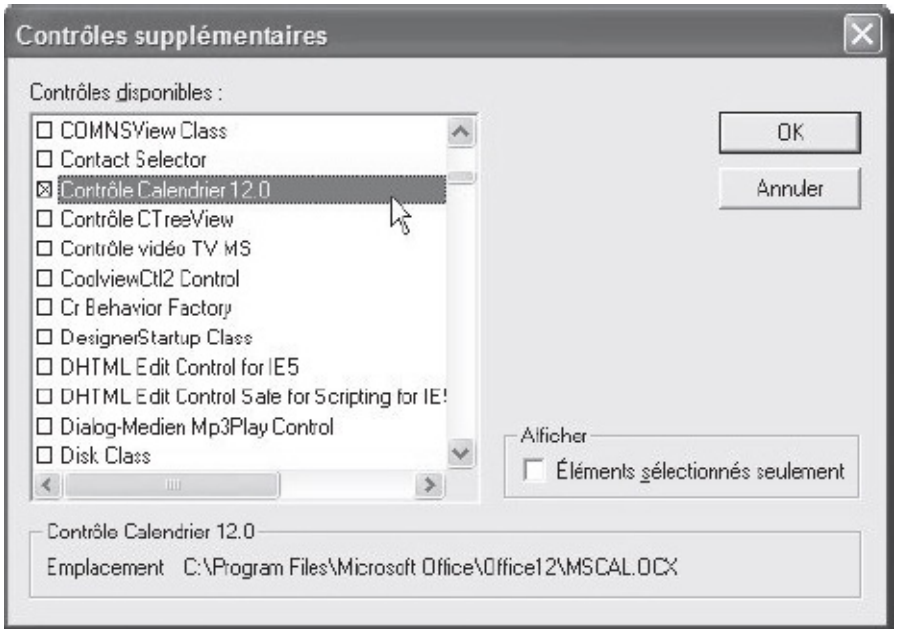


Figure 12-36 – *La boîte de dialogue Contrôles supplémentaires.*

2. Pour ajouter/supprimer un contrôle dans la boîte à outils, cochez/décochez sa case. Ici, cochez la case intitulée Calendar Control 11.0.
3. Cliquez sur OK. Le contrôle `Calendar` est maintenant disponible dans la boîte à outils (voir figure 12-37).

Figure 12-37 – *Le contrôle Calendar a été ajouté à la boîte à outils.*

4. Cliquez-droit sur l'icône de `Calendar` et sélectionnez Personnaliser un contrôle. La boîte de dialogue de la figure 12-38 s'affiche.

Figure 12-38 – *La boîte de dialogue Personnaliser un contrôle.*

5. Vous pouvez :
   – modifier le texte de l'info-bulle ;
   – modifier l'image représentant le contrôle sur la boîte à outils, en chargeant un fichier (bouton Charger une image) ou en la dessinant (Éditer une image).

Astuce

Pour supprimer des contrôles à partir de la boîte de dialogues Contrôles supplémentaires, cochez la case Afficher éléments sélectionnés seulement. Seuls les contrôles disponibles sur la boîte à outils seront proposés. Décochez alors ceux que vous souhaitez supprimer.

Vous pouvez aussi supprimer un contrôle à partir de la boîte à outils. Cliquez-droit dessus et sélectionnez la commande Supprimer contrôle.

La figure 12-39 présente une feuille sur laquelle un contrôle `Calendar` a été placé.

Ajouter/supprimer une page

Si vous exploitez de nombreux contrôles, il sera judicieux de les regrouper par catégories que vous placerez sur des pages distinctes de la boîte à outils.

Pour ajouter une page à la boîte à outils :

1. Cliquez sur l'onglet Contrôles de la boîte à outils et sélectionnez la commande Nouvelle page (voir figure 12-40). Par défaut, le libellé de son onglet est Nouvelle page et l'outil Sélection y est disponible.

Info

L'outil Sélection est toujours présent sur les pages de la boîte à outils et ne peut être supprimé.

Dates de remise et de parution
Indiquez la date à valider et sélectionnez-la sur le calendrier.
Date de remise :
09/12/2003
Date de parution :
27/12/2003
Date à valider :
Date de parution
December 2003
December
2003
lun. mar. mer. jeu. ven. sam. dim.
24 25 26 27 28 29 30
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31 1 2 3 4
Annuler
Retour
Terminer

Figure 12-39 – *Le contrôle Calendar permet à l'utilisateur de sélectionner une date sur un calendrier.*

Figure 12-40 – *Une nouvelle page ajoutée à la boîte à outils.*

2. Cliquez-droit sur l'onglet et choisissez la commande qui Renommer s'affiche (voir figure 12-41). Indiquez le nom de la page et, éventuellement, le texte d'info-bulle. Cliquez sur OK.

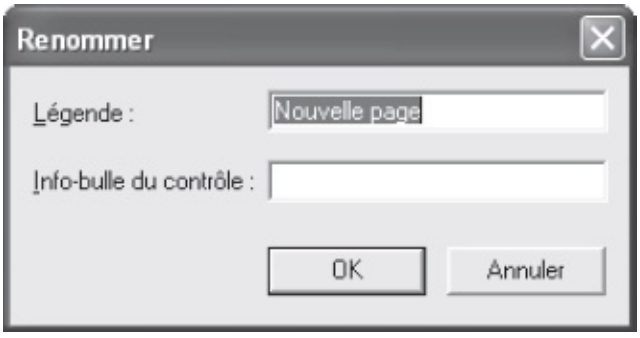


Figure 12-41 – *La boîte de dialogue Renommer.*

3. Pour ajouter ou supprimer des contrôles sur la nouvelle page, activez-la, puis procédez comme indiqué dans la section précédente.
4. Si vous souhaitez modifier l'emplacement de l'onglet sur la boîte à outils, cliquez dessus et sélectionnez la commande Déplacer. Utilisez les boutons Vers le haut et Vers le bas de la boîte de dialogue Ordre des pages (voir figure 12-42).

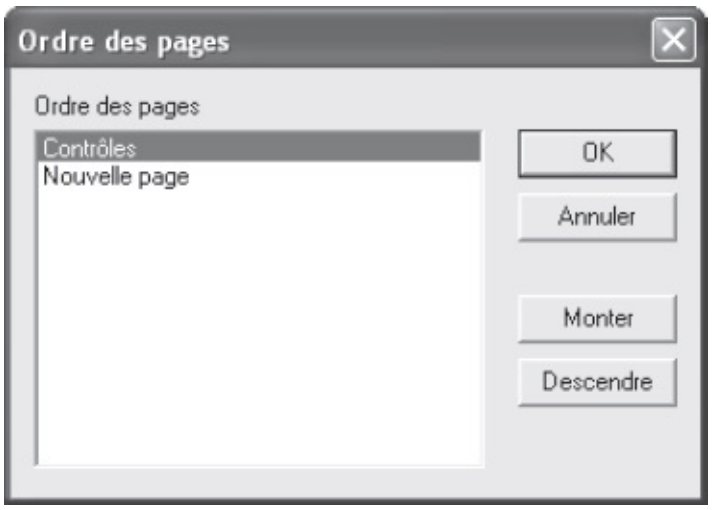


Figure 12-42 – *L'emplacement des onglets de la boîte à outils se détermine dans cette boîte de dialogue.*

Pour supprimer une page, cliquez-droit sur son onglet et choisissez la commande Supprimer la page.

Afficher/masquer une feuille

Une feuille ne peut s'exécuter d'elle-même. Son affichage doit être commandé à partir d'une procédure d'un module de code du projet. On utilise pour cela la méthode `Show`, selon la syntaxe suivante :

```
NomFeuille.Show
```

La procédure suivante affiche une boîte de dialogue invitant l'utilisateur à indiquer s'il souhaite que l'interface soit affichée. Si la réponse est positive, MaFeuille s'affiche. Dans le cas contraire, le programme se poursuit avec la procédure `SuiteDuProgramme`.

```
Sub AffichageFeuille()
  Dim RepAffichage As Integer
  RepAffichage = MsgBox("Souhaitez-vous afficher le formulaire ?", _
    vbYesNo + vbQuestion, "Affichage de feuille")
  If RepAffichage=vbYes Then
    MaFeuille.Show
  Else
    Call SuiteDuProgramme
  End If
End Sub
```

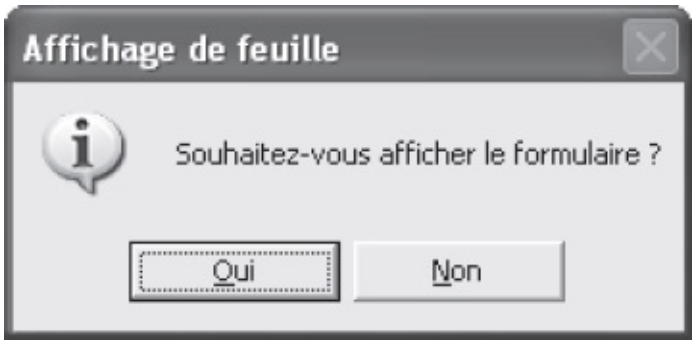


Figure 12-43 – *Si l'utilisateur clique sur Oui, la méthode Show sera appliquée à la feuille afin de l'afficher.*

Pour masquer une feuille, on lui applique la méthode `Hide`, selon la syntaxe suivante :

```
NomFeuille.Hide
```

Cette instruction est généralement placée dans la procédure événementielle affectée au bouton de validation de la feuille. Les valeurs entrées par l'utilisateur sont alors passées à une autre procédure qui les exploitera et la feuille est masquée. Dans l'exemple suivant, lorsque l'utilisateur clique sur le bouton cmdOK de `MaFeuille`, celle-ci est masquée et la procédure `SuiteDuProgramme` est appelée.

```
Private Sub cmdOK_Click()
  MaFeuille.Hide
  Call SuiteDuProgramme(arg1, arg2, …, argN)
End Sub
```

Astuce

La propriété `Me` renvoie la feuille active. Elle peut être utilisée dans n'importe quelle procédure événementielle attachée à une feuille pour la manipuler.

Si c'est la feuille active que vous souhaitez masquer (l'objet conteneur du contrôle ayant reçu l'événement), utilisez le mot-clé `Me` à la place de la propriété `Name` de la feuille (`Me.Hide`).

Info

L'affectation de procédures événementielles aux contrôles est traitée aux chapitres 13 et 14.

13

Exploiter les propriétés des contrôles

Au cours des chapitres précédents, vous avez découvert les différents contrôles disponibles et les outils destinés à leur mise en forme. Vous savez donc créer des formulaires à l'apparence professionnelle, mais ce ne sont pour l'instant que des feuilles sans vie. Ce chapitre et le suivant vous montrent comment tirer parti des feuilles VBA en exploitant les propriétés des contrôles et en associant du code aux événements les affectant.

En tant qu'objets, les feuilles et les contrôles possèdent un certain nombre de propriétés qui en déterminent l'apparence et le comportement. Il est fortement recommandé de préciser la propriété `Name` des contrôles au fur et à mesure que vous les placez sur la feuille.

Sélectionnez un objet et tapez sur la touche F4 afin d'en ouvrir la fenêtre Propriétés, ou cliquez-droit et sélectionnez la commande Propriétés du menu contextuel. Affectez ensuite les valeurs voulues à l'objet. Reportez-vous à la section « La fenêtre Propriétés » du chapitres 4.

Astuce

Si vous sélectionnez plusieurs contrôles sur la feuille, la fenêtre Propriétés affiche alors les propriétés qui leur sont communes. Les modifications que vous apporterez seront appliquées à tous les contrôles sélectionnés.

Il existe évidemment des propriétés spécifiques pour chaque contrôle et d'autres communes à presque tous. Les sections qui suivent vous présentent les propriétés essentielles, classées par catégories.

Dans la réalité, vous ne modifierez que très peu d'entre elles, car la valeur par défaut est le plus souvent satisfaisante. Les propriétés que vous devez connaître et que vous serez amené à utiliser présentent une icône en marge, identique à celle qui se trouve en marge de ce paragraphe.

Astuce

Sélectionnez le nom d'une propriété et tapez sur la touche F1 pour obtenir de l'aide. Vous pouvez aussi sélectionner le contrôle sur la feuille, puis taper sur F1 pour ouvrir la rubrique d'aide associée (voir figure 13-1).

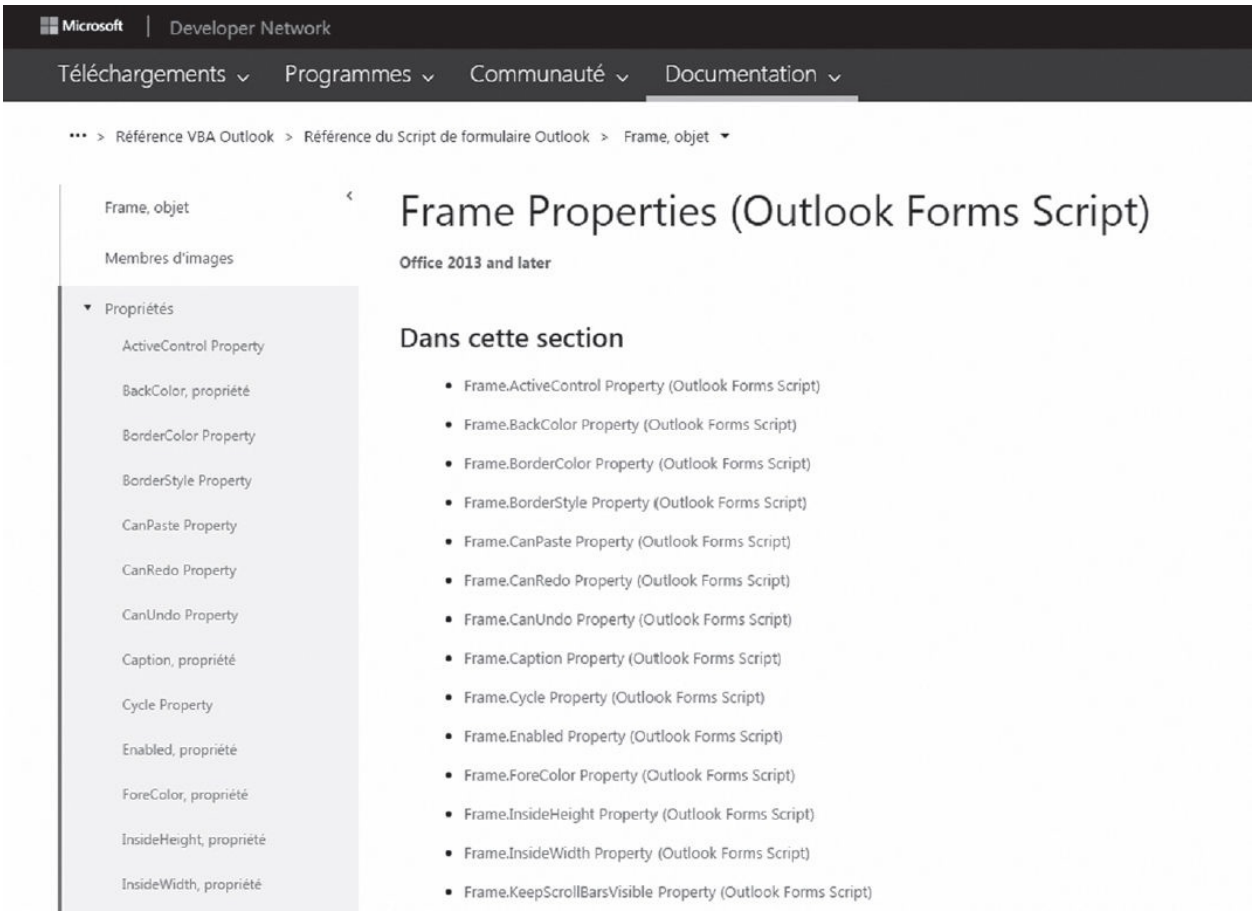


Figure 13-1 – *Accédez en un clin d'œil aux rubriques d'aide des propriétés d'un contrôle.*

Propriété Name

La propriété `Name` est exploitée par le programme et reste invisible à l'utilisateur final. Elle correspond au nom de l'objet et est de type `String` (chaîne de caractères). Ce nom est utilisé pour faire référence à l'objet dans le code, par exemple pour connaître ou modifier l'une de ses propriétés. Un même nom ne peut être utilisé pour plusieurs objets d'une même feuille.

Lorsque vous placez un contrôle sur une feuille, sa propriété `Name` est par défaut

une chaîne composée du nom de la classe de l'objet suivi d'un nombre entier. Par exemple, lorsque vous placez un bouton d'option, sa propriété `Name` est définie à `OptionButton1`, `OptionButton2` s'il existe déjà un contrôle `OptionButton1`, etc.

Conseil

Affectez des noms représentatifs aux contrôles de votre feuille. Il vous sera ainsi facile de savoir à quel contrôle fait référence une instruction dans le code.

L'instruction `If…End If` suivante teste la propriété `Value` d'une zone de texte (le texte saisi par l'utilisateur) dont la propriété `Name` est définie à `txtMotDePasse`. Si elle ne contient aucune information, un message demande à l'utilisateur d'entrer un mot de passe.

```
If txtMotDePasse.Value="" Then
  MsgBox "Vous devez indiquer un mot de passe.", VBOKOnly, "Mot de passe requis"
End If
```

Astuce

Si une zone de texte est destinée à recevoir un mot de passe, affectez un caractère tel que l'astérisque à sa propriété `PasswordChar`, afin qu'il s'affiche à la place des caractères saisis par l'utilisateur.

Vous devrez aussi protéger votre projet, afin que l'utilisateur ne puisse accéder au code de la procédure pour y lire le mot de passe : cliquez-droit sur le projet dans la fenêtre Propriétés et choisissez la commande Propriétés de `NomProjet`. Activez l'onglet Protection et cochez la case Verrouiller le projet pour l'affichage. Saisissez ensuite un mot de passe dans la zone de texte appropriée. Cette protection prendra effet lors du prochain lancement d'Excel.

Figure 13-2 – *Pour faire référence à un contrôle dans le code, utilisez sa propriété Name.*

Apparence

Alignment

Cette propriété détermine la position d'un contrôle `CheckBox` ou `OptionButton` par

rapport à sa légende (`Caption`). Elle prend pour valeur l'une des constantes `fmalignment` :

- **`fmalignmentright`** (par défaut) place le libellé à droite du contrôle.
- **`fmalignmentleft`** le place à gauche.

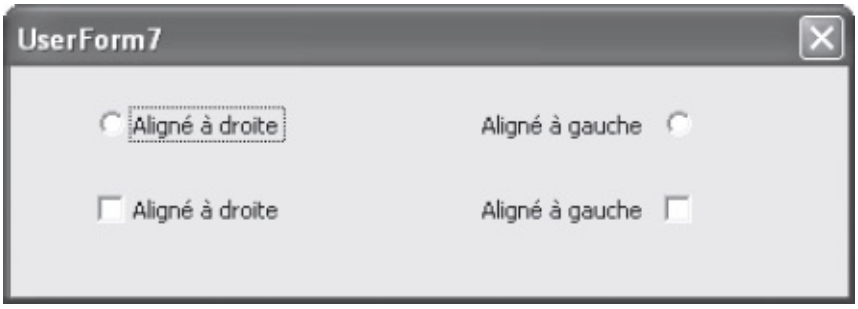


Figure 13-3 – *La légende d'un contrôle CheckBox ou RadioButton peut être alignée à droite ou à gauche du contrôle.*

BackStyle

Cette propriété détermine le style de fond du contrôle. Elle prend pour valeur l'une des constantes `fmBackStyle` :

- **`fmBackStyleOpaque`** (par défaut) détermine un fond opaque. Les éventuels objets placés à l'arrière-plan de l'objet sont invisibles.
- **`fmBackStyleTransparent`** détermine un fond transparent. Les éventuels objets placés à l'arrière-plan de l'objet sont visibles.

Color

Cette propriété prend pour valeur une variable de type `Long` représentant la couleur de fond du contrôle. Lorsque vous modifiez sa valeur dans la fenêtre Propriétés, vous pouvez sélectionner une couleur sur l'onglet Palette (voir figure 13-4) ou sur l'onglet Système.

Si la propriété `BackStyle` est définie sur `fmBackStyleTransparent`, `Color` est sans effet sur l'apparence du contrôle.

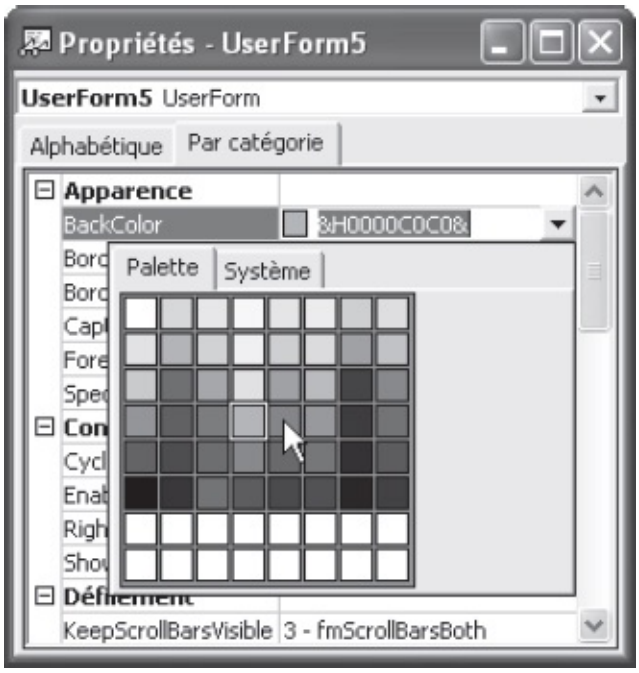


Figure 13-4 – *Sélectionnez une couleur de fond pour le contrôle dans cette palette.*

BorderStyle

Cette propriété détermine l'affichage ou non d'une bordure pour le contrôle. Elle prend pour valeur l'une des constantes `fmBorderStyle` :

- **`fmBorderStyleNone`** indique qu'aucune bordure ne s'affiche pour le contrôle.
- **`fmBorderStyleSingle`** indique qu'une bordure s'affiche.

BorderColor

Cette propriété détermine la couleur du contour du contrôle et peut prendre les mêmes valeurs que `BackColor`.

Si la propriété `BorderStyle` est définie sur `fmBorderStyleNone`, `BorderColor` est sans effet sur l'apparence du contrôle.

Caption

La propriété `Caption` détermine le texte descriptif d'un contrôle et accepte une

valeur de type `String`. Son emplacement varie selon le contrôle. Par exemple, le libellé d'un bouton de commande est centré sur le contrôle, tandis que celui d'une case à cocher ou d'un bouton d'option peut être aligné à droite (par défaut) ou à gauche, selon la valeur de la propriété `Alignment`. La figure 13-5 présente deux boutons de commande dont les propriétés `Caption` ont été définies à OK et à Annuler.

Figure 13-5 – *La propriété Caption sert à affecter une légende à un contrôle.*

Il peut être utile de faire varier la propriété `Caption` d'un contrôle en fonction des événements. Dans l'exemple suivant, lorsque l'utilisateur clique sur le bouton de commande dont `Name` est "JoueurSuivant", la propriété `Caption` de `MonLibellé` bascule entre "Joueur 1" et "Joueur 2".

```
Private Sub JoueurSuivant_Click()
  Call AutreProcédure
  If MonLibellé.Caption="Joueur 1" Then
    MonLibellé.Caption = "Joueur 2"
  Else
    MonLibellé.Caption = "Joueur 1"
  End If
End Sub


Sub AutreProcédure()
  Instructions
End Sub
```

La première ligne appelle une procédure d'événement (traitée au chapitre suivant) lorsque l'utilisateur clique sur le bouton dont la propriété `Name` est "JoueurSuivant". Les *`Instructions`* de `AutreProcédure` s'exécutent, puis la procédure appelante reprend la main. Enfin, l'instruction `If…End If` modifie la propriété `Caption` de `MonLibellé` : si elle vaut "Joueur 1", elle est redéfinie à "Joueur 2" et inversement.

Attention

Ne confondez pas le libellé d'un contrôle (`Caption`) et son nom (`Name`). Le nom sert à faire référence à ce contrôle dans le code du programme et est invisible pour l'utilisateur final.

ControlTipText

Cette propriété détermine le texte de l'info-bulle qui s'affiche lorsque l'utilisateur place le pointeur au-dessus du contrôle sans cliquer. Elle accepte une valeur de type `String` et contient par défaut une chaîne vide. N'hésitez pas à utiliser cette propriété pour décrire brièvement à l'utilisateur la fonction des contrôles d'une feuille (figure 13-6).

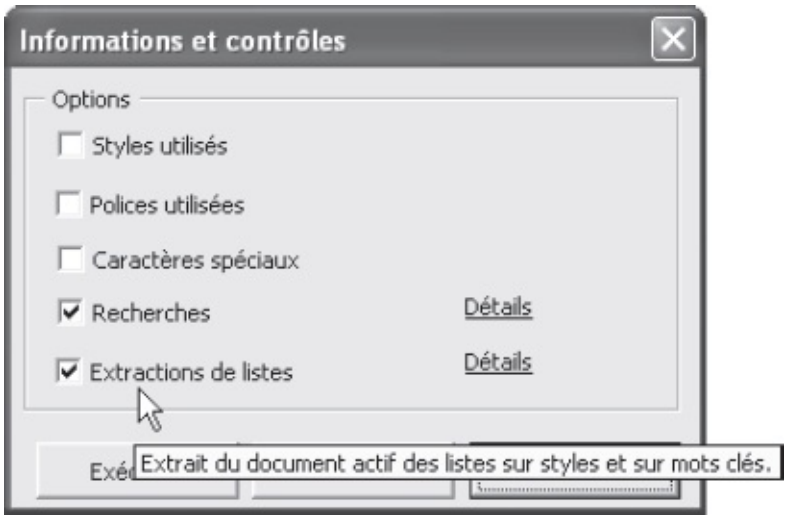


Figure 13-6 – *Les info-bulles renseignent l'utilisateur sur la fonction d'un contrôle.*

ForeColor

Cette propriété détermine la couleur de premier plan du contrôle, pour l'affichage de sa propriété `Caption`.

SpecialEffect

Cette propriété détermine l'apparence du contrôle sur la feuille :

- Les contrôles `CheckBox`, `RadioButton` et `ToggleButton` acceptent l'une des constantes suivantes :
  - `fmButtonEffectFalt` ;
  - `fmButtonEffectSunken`.
- Les autres contrôles acceptent l'une des constantes suivantes :

- `fmSpecialEffectFlat` ;
- `fmSpecialEffectRaised` ;
- `fmSpecialEffectSunken` ;
- `fmSpecialEffectEtched`.

La valeur attribuée par défaut à cette propriété correspond à l'apparence habituelle de ce type d'objets dans les applications Windows.

Style

Cette propriété détermine la façon dont la sélection s'effectue dans un contrôle `ComboBox`: `1`'utilisateur peut ou non être autorisé à saisir une valeur ne figurant pas dans la liste. Elle accepte pour valeur l'une des deux constantes `fmStyle` suivantes :

- **`fmStyleDropDownCombo`** (par défaut). L'utilisateur peut sélectionner un item dans la liste déroulante ou saisir manuellement une valeur de son choix. C'est, par exemple, le cas de la liste Zoom de la barre d'outils Standard d'Excel.
- **`fmStyleDropDownList`**. Les seules valeurs autorisées sont celles de la liste attachée au contrôle. Lorsque l'utilisateur saisit un caractère au clavier, le premier élément de la liste commençant par ce caractère est sélectionné. Si aucun élément de la liste ne correspond, la valeur du contrôle reste inchangée.

Utilisez un contrôle `ComboBox` dont la propriété `Style` est définie à `fmStyleDropDownList`, plutôt qu'un `ListBox`, pour réduire l'espace occupé sur la feuille.

Attention

Lorsque vous autorisez la saisie manuelle dans la zone d'édition d'un contrôle `ComboBox`, assurez-vous que l'utilisateur ne pourra y entrer une valeur non valide, qui générerait une erreur ou aboutirait à un mauvais fonctionnement du programme. Affectez pour cela une procédure à l'événement `Change` du contrôle, qui contrôlera la valeur saisie (voir l'exemple donné pour le contrôle `TextBox`, au chapitre suivant).

Value

Cette propriété détermine la valeur d'un contrôle, qui peut correspondre à un état (le fait qu'une case soit ou non cochée : valeur numérique) ou à un contenu (le texte saisi dans la zone d'édition d'une zone de texte : type `String`).

Le tableau suivant présente les valeurs acceptées par la propriété `Value` des contrôles standards.

Contrôle	Valeur acceptée par la propriété `Value`
`CheckBox`, `OptionButton` ou `ToggleButton`	Valeur de type `Boolean`, indiquant l'état du contrôle : `True` s'il est activé et `False` s'il est désactivé. Les contrôles `CheckBox` et `OptionButton` sont activés lorsqu'ils sont cochés ; un contrôle `ToggleButton` l'est lorsqu'il est en surbrillance et enfoncé. Si leur propriété `TripleState` est définie à `True`, ces contrôles acceptent un troisième état : ni activé ni inactivé. `Value` prend alors la valeur `Null`. Une case à cocher ou un bouton d'option apparaît alors estompé et contient une marque d'activation. Lorsqu'un bouton bascule est à l'état `Null`, son libellé apparaît estompé. *Voir aussi* la propriété `TripleState`.
`TextBox`	Valeur de type `String` représentant le texte qui apparaît dans la zone d'édition du contrôle.
`ComboBox` et `ListBox`	Valeur représentant l'élément sélectionné dans la liste. La propriété `BoundColumn` en détermine le type. Il peut s'agir de la *valeur d'index* de l'élément sélectionné (sa position dans la liste) ou d'une chaîne correspondant au texte sélectionné. Dans le cas d'un contrôle à colonnes multiples, il peut s'agir du texte de n'importe laquelle des colonnes. La propriété `Value` n'est pas utilisable pour une liste à sélection multiple (propriété `MultiSelect`). Pour plus de précisions, reportez-vous aux sections traitant de ces contrôles, plus loin dans ce chapitre. *Voir aussi* la propriété `BoundColumn`.
`ScrollBar` et `SpinButton`	Valeur de type `Integer` comprise entre le `Min` et le `Max` du contrôle. Pour un `ScrollBar`, elle indique la position du curseur sur la barre de défilement : `Min` lorsqu'il est tout en haut (barre verticale) ou tout à gauche (barre horizontale) et `Max` lorsqu'il est tout en bas ou à droite. Pour un `SpinButton`, elle représente la valeur sélectionnée par un clic sur les boutons du contrôle ; elle se rapproche de `Min` lorsque l'utilisateur clique sur la flèche Haut ou Gauche et de `Max` lorsqu'il clique sur la flèche Bas ou Droite. *Voir aussi* les propriétés `Min` et `Max`.
`MultiPage`	Valeur de type `Integer` représentant la page sélectionnée, comprise entre 0 et le nombre de pages moins 1.

La procédure suivante se déclenche lorsque l'utilisateur clique sur le bouton de commande btnOK. Elle teste la valeur de la propriété `Value` du contrôle txtMotdePasse, une zone de texte dans laquelle l'utilisateur est invité à saisir son mot de passe. Si elle est égale à une chaîne vide, la boîte de dialogue de la figure 13-7 s'affiche.

```
If txtMotDePasse.Value="" Then
  MsgBox "Vous devez indiquer un mot de passe.", VBOKOnly, "Mot de passe requis"
```

```
Else
  Instructions
End If
```

Figure 13-7 – *Value est l'une des propriétés les plus utilisées.*

Visible

Cette propriété détermine la visibilité d'un contrôle pour l'utilisateur. Elle accepte une valeur de type `Boolean` :

- **`True`** (par défaut) indique que le contrôle est visible.
- **`False`** indique que le contrôle est masqué et n'apparaît pas sur la feuille.

> **Info**
> La propriété `Visible` d'un contrôle n'influe pas sur son apparence durant la phase de conception. Le contrôle est toujours visible dans Visual Basic Editor.

La propriété `Visible` permet d'élaborer des feuilles dont l'apparence et les fonctionnalités varieront selon le contexte. Vous pouvez ainsi n'afficher les contrôles d'une feuille que s'ils doivent être renseignés par l'utilisateur. La figure 13-8 présente la feuille Nouveau salarié, en phase de conception dans Visual Basic Editor.

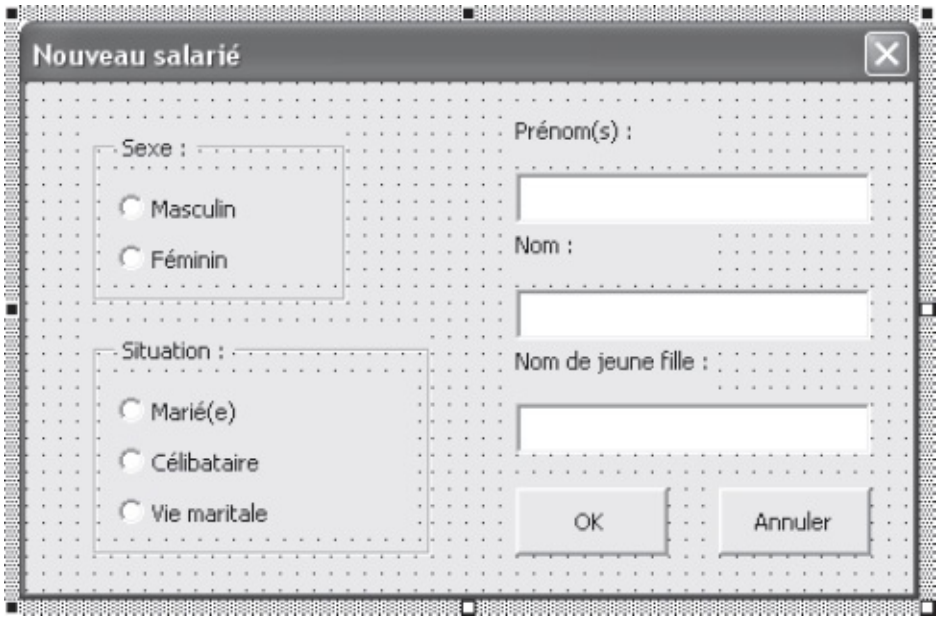


Figure 13-8 – *La feuille Nouveau salarié en phase de conception.*

Outre le sexe et la situation, l'utilisateur est invité à entrer un prénom et un nom. Une zone de texte sert à indiquer un éventuel nom de jeune fille.

Les propriétés des contrôles `OptionButton`, `CheckBox` et `Label` de la feuille ont été définies comme suit :

Propriété *Name*	Propriété *Caption*
Contrôles *OptionButton*	
`optMasculin`	Masculin
`optFéminin`	Féminin
`optMarié`	Marié(e)
`optCélibataire`	Célibataire
`optMaritale`	Vie maritale
Contrôles *Label*	
`lbPrénom`	Prénom(s) :
`lbNom`	Nom :
`lbNomJF`	Nom de jeune fille :
Contrôles *TextBox*[1]	
`txtPrénom`	
`txtNom`	
`txtNomJF`	

1. Les contrôles `TextBox` n'ont pas de propriété `Caption`.

Il est évidemment inutile d'afficher le libellé Nom de jeune fille et la zone de texte correspondante si les boutons d'option Féminin et Marié(e) ne sont pas tous deux cochés. Leur propriété `Visible` a donc été définie à `False` – la propriété `Value` du contrôle `optMasculin` étant définie à `True` afin que ce bouton soit coché par défaut. Le code suivant affiche les contrôles `txtNomJF` et `lbJF` lorsque cela est nécessaire et les masque dans le cas contraire.

```
Private Sub optFéminin_Click()
  If optMarié.Value=True Then
    txtNomJF.Visible = True
    lbJF.Visible = True
  Else
    txtNomJF.Visible = False
    lbJF.Visible = False
  End If
End Sub


Private Sub optMarié_Click()
  If optFéminin.Value=True Then
    txtNomJF.Visible = True
    lbJF.Visible = True
  Else
    txtNomJF.Visible = False
    lbJF.Visible = False
  End If
End Sub


Private Sub optMasculin_Click()
  txtNomJF.Visible = False
  lbJF.Visible = False
End Sub


Private Sub optCélibataire_Click()
  txtNomJF.Visible = False
  lbJF.Visible = False
End Sub


Private Sub optMaritale_Click()
  txtNomJF.Visible = False
  lbJF.Visible = False
End Sub
```

La procédure `optFéminin_Click` se déclenche lorsque l'utilisateur clique sur le contrôle nommé `optFéminin`. Si la propriété `Value` du contrôle `optMarié` est définie à `True`, la propriété `Visible` des contrôles `txtNomJF` et `lbJF` est définie à `True` pour les faire apparaître sur la feuille.

La procédure `optMarié_Click` se déclenche lorsque l'utilisateur clique sur le contrôle nommé `optMarié` et opère de façon semblable.

Dans les autres cas, `txtNomJF` et `lbJF` restent invisibles.

Figure 13-9 – *La propriété Visible permet de déterminer les conditions d'apparition des contrôles pour l'utilisateur.*

Comportement

AutoSize

Cette propriété spécifie si un contrôle se redimensionne automatiquement pour afficher son contenu, c'est-à-dire la valeur de sa propriété `Caption` ou, dans le cas d'un `TextBox` ou d'un `ComboBox`, le texte qui apparaît dans sa zone d'édition. La propriété `AutoSize` accepte une valeur de type `Boolean` :

- **`True`**. Le contrôle se redimensionne automatiquement.
- **`False`** (par défaut). La taille du contrôle est fixe.

> **Info**
>
> Lorsque vous définissez la propriété `AutoSize` d'un contrôle durant la phase de conception, il se redimensionne automatiquement en fonction de son contenu. Si vous le souhaitez, définissez une taille supérieure. En phase d'exécution, le contrôle conservera la taille ainsi définie, tant que celle-ci suffira à afficher tout le contenu.

La figure 13-10 présente deux `TextBox` dont les propriétés `AutoSize` sont respectivement définies sur `True` et `False`. L'un s'est redimensionné automatiquement, tandis que le texte du second est rogné.

Figure 13-10 – *La propriété AutoSize évite que le texte ne soit rogné.*

Conseil

Ne définissez la propriété `AutoSize` d'un contrôle `TextBox` sur `True` que si le nombre de caractères autorisés dans ce contrôle (propriété `MaxLength`) est limité. Dans le cas contraire, l'utilisateur pourrait saisir un nombre illimité de caractères et le contrôle dépasserait les limites de la feuille.

AutoTab

Disponible pour les contrôles `TextBox` et `ComboBox` dont la propriété `MaxLength` a été définie, `AutoTab` spécifie si une tabulation automatique se déclenche lorsque le nombre maximal de caractères est atteint. Si tel est le cas, le ***focus*** est passé au contrôle suivant dans l'***ordre de tabulation***.

Info

Le contrôle qui a le focus est réceptif aux événements utilisateur (souris ou clavier). Un seul peut avoir le focus à un moment donné et le type d'événements reconnus varie d'un contrôle à l'autre.

Par exemple, taper sur la touche Entrée alors qu'un bouton de commande a le focus revient à cliquer sur ce bouton. Cela n'aura en revanche aucun effet si le focus correspond à une case à cocher, alors que chaque frappe de la touche Tab la fera passer d'un état à l'autre.

Définition

L'ordre de tabulation désigne l'ordre dans lequel le focus est transmis aux différents contrôles d'une feuille et est déterminé par leur propriété `TabIndex`.

`AutoTab` accepte une valeur de type `Boolean` :

- **`True`**. Le focus est passé au contrôle suivant dans l'ordre de tabulation lorsque le nombre de caractères maximal est atteint.
- **`False`** (par défaut). Lorsque le nombre de caractères maximal est atteint, les événements clavier demeurent sans effet, mais le contrôle `TextBox` garde le focus.

La propriété `AutoTab` sera utilisée judicieusement avec une zone de texte devant toujours recevoir un nombre fixe de caractères, comme un mot de passe ou un code produit. Le contrôle suivant sera ainsi sélectionné automatiquement sans que l'utilisateur ait à quitter le clavier.

AutoWordSelect

Cette propriété détermine la façon dont le texte est sélectionné dans la zone d'édition d'un contrôle `TextBox` ou `ComboBox`. L'unité de sélection peut être le mot ou le caractère. `AutoWordSelect` accepte une valeur de type `Boolean` :

- **`True`** (par défaut). La sélection se fait mot par mot au-delà du premier mot.
- **`False`**. La sélection se fait caractère par caractère, quelle que soit son étendue.

Cancel

Cette propriété spécifie si un contrôle `CommandButton` est ou non le bouton Annuler de la feuille. Lorsque c'est le cas, la frappe de la touche Échap déclenche l'événement `Click` du bouton, qu'il ait ou non le focus. Outre cet accès, le bouton est toujours activable par les méthodes classiques – raccourci clavier, clic de souris ou touche Entrée lorsque le contrôle a le focus. `Cancel` accepte une valeur de type `Boolean` :

- **`True`**. Le bouton de commande est le bouton Annuler de la feuille.
- **`False`** (par défaut). Il n'est pas le bouton Annuler.

Dans le cas d'une procédure complexe, ou nécessitant un grand nombre d'opérations de la part de l'utilisateur, protégez l'activation du bouton d'annulation en affichant une boîte de dialogue demandant la confirmation de l'abandon.

La procédure suivante affiche la boîte de dialogue présentée à la figure 13-11 lorsque l'utilisateur clique sur le bouton de commande cmdAnnuler.

```
Private Sub cmdAnnuler_Click()
  Dim Confirm As Single
  Confirm = MsgBox("Si vous annulez la procédure, vous perdrez toutes les données
    ➥ entrées. " _
    & "Etes-vous sûr de vouloir annuler ?", vbYesNo + vbCritical, "Abandon de la
    ➥ procédure")
  If Confirm=vbYes Then
    Exit Sub
  End If
End Sub
```

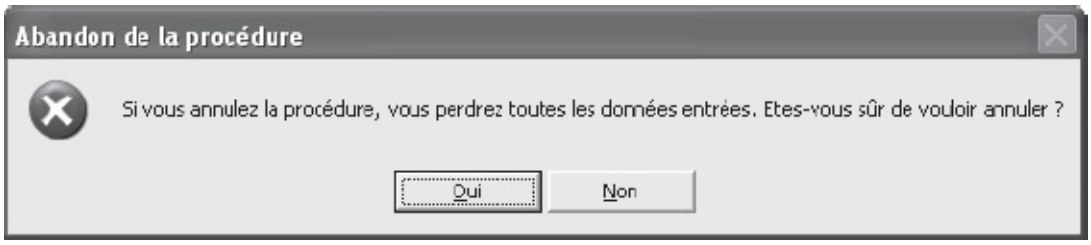

Abandon de la procédure
Si vous annulez la procédure, vous perdrez toutes les données entrées. Etes-vous sûr de vouloir annuler ?
Oui
Non

Figure 13-11 – *Protégez l'utilisateur d'un abandon involontaire de la procédure par l'affichage d'un message de confirmation.*

Info

Sur une même feuille, il ne peut y avoir qu'un seul bouton Annuler. Lorsque vous définissez la propriété `Cancel` d'un bouton de commande sur `True`, elle est automatiquement définie à `False` pour tous les autres.

Default

Cette propriété spécifie si un contrôle `CommandButton` est ou non le bouton par défaut de la feuille. Lorsque c'est le cas, la frappe de la touche Entrée déclenche l'événement `Click` du bouton, à condition qu'il ait le focus. Outre cet accès, le bouton est toujours activable par les méthodes classiques – raccourci clavier, clic de souris. `Default` accepte une valeur de type `Boolean` :

- **`True`.** Le bouton de commande est celui par défaut.
- **`False`** (par défaut). Il n'est pas le bouton par défaut.

Info

Sur une même feuille, il ne peut y avoir qu'un seul bouton par défaut. Lorsque vous définissez la propriété `Default` d'un bouton de commande sur `True`, elle est automatiquement définie à `False` pour tous les autres.

Enabled

Cette propriété détermine si un contrôle est ***accessible***, c'est-à-dire qu'il peut avoir le focus, ou ***inaccessible***, auquel cas il apparaît estompé et l'utilisateur ne peut le sélectionner ni à l'aide de la souris ni à l'aide de la touche Tab. `Enabled` accepte une valeur de type `Boolean` :

- **`True`** (par défaut). Le contrôle est accessible.
- **`False`**. Il n'est pas accessible.

Utilisez cette propriété pour déterminer l'accessibilité d'un contrôle en fonction du contexte. Dans l'exemple suivant, nous avons ajouté un `CheckBox` sur la feuille, afin d'indiquer si le nouveau salarié est dégagé ou non des obligations militaires – sa propriété `Name` a été définie à `chkMilitaire`. Il est inutile que l'utilisateur accède à ce contrôle si le nouveau salarié est une femme. Nous avons ajouté deux instructions dans ce sens.

```
Private Sub optFéminin_Click()
  If optMarié.Value=True Then
    txtNomJF.Visible = True
    lbJF.Visible = True
  Else
    txtNomJF.Visible = False
    lbJF.Visible = False
  End If
  chkMilitaire.Enabled = False
End Sub
Private Sub optMarié_Click()
  If optFéminin.Value=True Then
    txtNomJF.Visible = True
    lbJF.Visible = True
  Else
    txtNomJF.Visible = False
    lbJF.Visible = False
  End If
End Sub


Private Sub optMasculin_Click()
  txtNomJF.Visible = False
  lbJF.Visible = False
  chkMilitaire.Enabled = True
End Sub


Private Sub optCélibataire_Click()
  txtNomJF.Visible = False
  lbJF.Visible = False
End Sub


Private Sub optMaritale_Click()
  txtNomJF.Visible = False
  lbJF.Visible = False
```

```
End Sub
```

Info

Notez qu'un contrôle peut être accessible et non modifiable. Pour plus de précisions, voyez la propriété `Locked`.

Figure 13-12 – *La propriété Enabled détermine l'accessibilité d'un contrôle en fonction du contexte.*

EnterKeyBehavior

Cette propriété détermine le comportement d'un contrôle `CheckBox` lorsque l'utilisateur appuie sur la touche Entrée. Elle accepte une valeur de type `Boolean` :

- **`True`**. Une pression sur la touche Entrée crée une nouvelle ligne.
- **`False`** (par défaut). Une pression sur la touche Entrée passe le focus au contrôle suivant dans l'ordre de tabulation.

Info

Si la propriété `MultiLine` d'un `CheckBox` est définie sur `False`, le contrôle n'autorise qu'une ligne et une pression sur la touche Entrée entraînera toujours le passage du focus au contrôle suivant dans l'ordre de tabulation.

HideSelection

Cette propriété détermine si la sélection dans un `TextBox` ou un `ComboBox` demeure ou non visible lorsque le contrôle n'a pas le focus. Elle accepte une valeur de type `Boolean` :

- **`True`** (par défaut). La sélection n'est visible que lorsque le contrôle a le focus.
- **`False`**. La sélection demeure visible, que le contrôle ait le focus ou non.

Maintenir la sélection visible dans un `TextBox` ou un `ComboBox` lorsque celui-ci n'a pas le focus est déroutant pour l'utilisateur, car cette propriété est généralement spécifique du contrôle ayant le focus.

Locked

Cette propriété détermine si un contrôle peut ou non être modifié ou activé par l'utilisateur. Elle accepte une valeur de type `Boolean` :

- **`True`**. Le contrôle peut être modifié.
- **`False`** (par défaut). Il n'est pas modifiable.

Conseil

Utilisez `Locked` conjointement avec `Enabled`, par exemple pour laisser l'utilisateur copier le contenu d'une zone de texte sans l'autoriser à le modifier.

`Locked` doit être définie sur `True` pour un contrôle ayant une fonction de consultation ; par exemple, un `TextBox` affichant le texte sélectionné dans le document ou le résultat d'un calcul.

Info

Lorsque la propriété `Locked` est définie à `True`, le contenu du contrôle n'est pas modifiable directement sur la feuille durant la phase Conception. Si vous souhaitez modifier sa propriété `Caption` ou `Value`, vous devez le faire dans la fenêtre Propriétés.

MaxLength

Cette propriété spécifie le nombre maximal de caractères que l'utilisateur est autorisé à saisir dans un `TextBox` ou un `ComboBox`. Elle accepte une valeur de type `Integer`. Par défaut, elle est définie sur 0, l'utilisateur étant alors autorisé à entrer un nombre de caractères indéterminé.

Lorsque le nombre maximal de caractères est atteint, les frappes de clavier restent sans effet. Vous pouvez aussi choisir que le focus soit passé au contrôle suivant dans l'ordre de tabulation en définissant la propriété `AutoTab` sur `True`.

> **Conseil**
> Affectez une valeur `MaxLength` aux `TextBox` destinés à recevoir un nombre de caractères constant (mot de passe ou code produit, par exemple), ou lorsque la limitation est une nécessité du programme.

Voir aussi `AutoTab`.

MultiLine

Cette propriété détermine si le texte saisi dans un `TextBox` s'affiche sur la ligne suivante lorsqu'il arrive en bout de ligne. Si tel est le cas, le texte s'affiche sur la ligne suivante tant que les dimensions du contrôle le permettent. `MultiLine` accepte une valeur de type `Boolean` :

- **`True`** (par défaut). Le texte s'affiche sur plusieurs lignes lorsque cela est nécessaire.
- **`False`**. Il s'affiche sur une seule ligne.

SelectionMargin

Cette propriété détermine si l'utilisateur peut ou non sélectionner une ligne de texte dans un `TextBox` ou un `ComboBox` avec le curseur. Si tel est le cas, le curseur se transforme en flèche lorsqu'il est placé à gauche de la ligne à sélectionner (voir figure 13-13). `SelectionMargin` accepte une valeur de type `Boolean` :

- **`True`** (par défaut). Ce type de sélection est possible.
- **`False`**. Ce type de sélection est impossible.

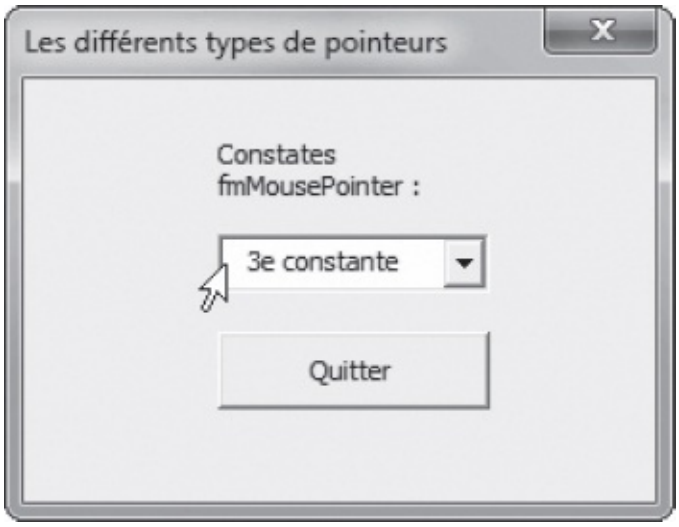


Figure 13-13 – *Le curseur prend la forme d'une flèche indiquant qu'un clic entraînera la sélection de la ligne entière.*

Style

Cette propriété détermine si un `ComboBox` autorise ou non la saisie d'une valeur ne figurant pas dans sa liste déroulante. `Style` accepte pour valeur une constante `fmStyle` :

- **`fmStyleDropDownCombo`** (par défaut). Le contrôle se comporte comme une zone de texte. L'utilisateur peut saisir une valeur de son choix dans la zone d'édition ou sélectionner l'une des options de la liste déroulante. Si les premières lettres saisies sont rapprochées de l'une de ces options, le complément automatique est proposé. L'utilisateur doit alors accepter le complément proposé ou saisir une valeur différente.
- **`fmStyleDropDownList`**. L'utilisateur ne peut sélectionner qu'une des options de la liste. Il est impossible de saisir une valeur différente dans la zone d'édition du contrôle. Si l'utilisateur saisit une lettre au clavier alors que le contrôle a le focus, l'option la plus proche de la lettre saisie est sélectionnée.

Info

La façon dont le rapprochement est effectué entre la valeur saisie par l'utilisateur dans la zone d'édition et les options de la liste est déterminée par les propriétés `MatchEntry` et `MatchRequired` du contrôle.

TabKeyBehavior

Cette propriété détermine si l'utilisateur peut ou non saisir une tabulation dans

la zone d'édition d'un `TextBox`. Elle accepte une valeur de type `Boolean` :

- **`True`**. L'utilisateur peut saisir une tabulation dans la zone d'édition. La touche Tab n'est donc pas utilisable pour passer le focus au contrôle suivant dans l'ordre de tabulation.
- **`False`** (par défaut). La touche Tab entraîne le passage du focus au contrôle suivant.

> **Conseil**
> Ne définissez cette propriété sur `True` que si l'exploitation des données du contrôle par le programme prévoit la gestion des tabulations.

TextAlign

Cette propriété détermine la façon dont est aligné le libellé d'un `Label` ou le texte dans un `TextBox` ou un `ComboBox`. Elle accepte l'une des trois constantes `fmTextAlign` :

- **`fmTextAlignLeft`** (par défaut). Aligné à gauche.
- **`fmTextAlignCenter`**. Centré.
- **`fmTextAlignRight`**. Aligné à droite.

> **Info**
> Dans le cas d'un `ComboBox`, `TextAlign` ne détermine que l'alignement du texte saisi dans la zone d'édition et non les entrées de la liste, qui sont toujours alignées à gauche.

TripleState

Cette propriété détermine si un contrôle `OptionButton`, `CheckBox` ou `ToggleButton` autorise ou non l'état `Null` (ni activé ni désactivé). Une case à cocher ou un bouton d'option à l'état `Null` apparaît estompé et contient une marque d'activation (voir figure 13-14). Sa propriété `Value` renvoie alors la valeur `Null`. Lorsqu'un bouton bascule à l'état `Null`, son libellé apparaît estompé. `TripleState` accepte une valeur de type `Boolean` :

- **`True`**. Le contrôle accepte l'état `Null`.
- **`False`** (par défaut). Le contrôle n'accepte que les états activé et désactivé.

Figure 13-14 – *Un contrôle CheckBox peut accepter trois états.*

Voir aussi `Value`.

WordWrap

Cette propriété détermine si un contrôle autorise un changement de ligne afin d'afficher la totalité de son contenu. Elle est gérée par les contrôles possédant une propriété `Caption` et par `TextBox`, mais pas par `ListBox` ni `ComboBox`. `WordWrap` accepte une valeur de type `Boolean` :

- **`True` (par défaut, sauf pour les `CommandButton`).** Le contrôle autorise les changements de ligne.
- **`False`.** Il ne les autorise pas.

Lorsqu'un contrôle autorise un changement de ligne, celui-ci s'opère lorsque le contenu atteint la limite d'affichage de ligne et si la hauteur autorise la création d'une ligne supplémentaire. Si tel n'est pas le cas, aucun changement de ligne ne s'effectue. La figure 13-15 présente deux `TextBox` dans lesquels un même texte a été saisi. La propriété `WordWrap` du premier est définie à `True` et celle du second à `False`.

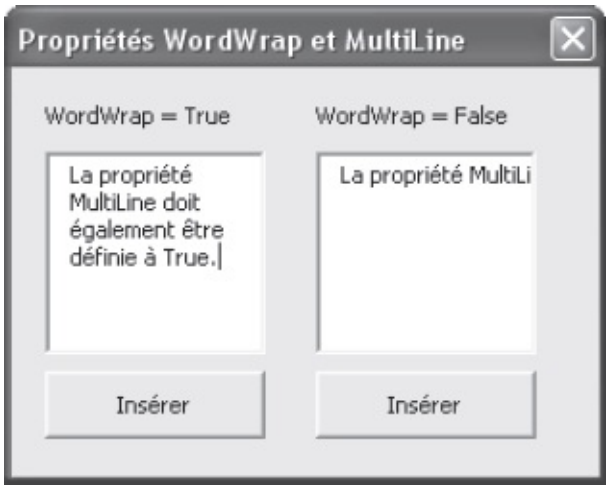


Figure 13-15 – *La propriété WordWrap permet de visualiser la totalité du contenu du contrôle, en autorisant l'affichage sur plusieurs lignes.*

Info

Si la propriété `MultiLine` d'un `TextBox` est définie sur `False`, une seule ligne est autorisée dans la zone d'édition ; `WordWrap` est donc sans effet.

Défilement

ScrollBars

Cette propriété détermine l'affichage d'une ou de plusieurs barres de défilement pour les feuilles, les `TextBox` et les `Frame`. Les barres apparaissent lorsque les dimensions du contrôle ne permettent pas d'afficher la totalité de son contenu. `ScrollBars` accepte l'une des constantes `fmScrollBars` :

- **`fmScrollBarsNone`.** Aucune barre de défilement ne s'affiche.
- **`fmScrollBarsHorizontal`.** Le contrôle possède une barre de défilement horizontale.
- **`fmScrollBarsVertical`.** Il possède une barre verticale.
- **`fmScrollBarsBoth`.** Il possède une barre horizontale et une verticale.

Info

Les `ListBox` ne gèrent pas la propriété `ScrollBars`. Ils affichent toujours une barre de défilement verticale lorsque cela est nécessaire.

Un `TextBox` dont la propriété `MultiLine` est définie à `False` ne peut présenter de barre verticale, puisque son contenu est toujours affiché sur une seule ligne. Un `TextBox` dont la propriété `WordWrap` est définie à `True` ne peut posséder de barre horizontale, puisque, chaque fois que la limite d'affichage liée aux dimensions du contrôle est atteinte, le contenu passe à la ligne suivante.

Une barre de défilement permet d'afficher une zone supérieure à la taille réelle d'une feuille ou d'un contrôle `MultiPage`. Vous devez donc spécifier la taille totale de la zone affichable à l'aide des barres de défilement (propriétés `ScrollHeight` et `ScrollWidth`). Elle doit évidemment être supérieure à la taille réelle du contrôle (`Height`). `ScrollHeight` est, par conséquent, généralement définie dans une procédure, relativement à la valeur de `Height` (par exemple, `MonContrôle.ScrollHeight = MonContrôle.Height * 1.5`).

Pour plus de précisions sur l'utilisation des barres de défilement sur les feuilles et les contrôles `MultiPage`, reportez-vous à l'Aide Visual Basic.

Conseil

Lorsque vous définissez à `True` les propriétés `MultiLine` et `WordWrap` d'un `TextBox`, affectez la valeur `fmScrollBarsVertical` à sa propriété `ScrollBars`. Une barre de défilement sera ainsi ajoutée lorsque les capacités d'affichage du contrôle seront dépassées.

Conseil

Lorsque vous affectez une barre de défilement à un contrôle, assurez-vous que ce dernier dispose d'assez d'espace sur la feuille pour l'afficher.

KeepScrollsVisible

Cette propriété détermine si les barres de défilement d'un contrôle sont ou non visibles lorsqu'elles n'ont aucune utilité, c'est-à-dire lorsque la totalité du contenu du contrôle s'affiche. Elle accepte l'une des constantes `fmScrollBars` :

- **`fmScrollBarsNone`.** Les barres de défilement ne sont affichées que lorsque les dimensions du contrôle ne permettent pas d'en visualiser le contenu.
- **`fmScrollBarsHorizontal`.** La barre horizontale est toujours affichée.
- **`fmScrollBarsVertical`.** La barre verticale est toujours affichée.
- **`fmScrollBarsBoth`.** Les barres horizontale et verticale sont toujours affichées.

Delay

Cette propriété détermine la périodicité avec laquelle les événements `SpinUp`, `SpinDown` et `Change` sont reconnus sur un contrôle `SpinButton` ou `ScrollBar`, lorsque l'utilisateur clique sur les flèches ou dans la barre de défilement et maintient le bouton enfoncé. Elle accepte une valeur numérique de type `Long`, qui représente la périodicité de déclenchement de l'événement en millisecondes. Sa valeur par défaut est 50.

Pour vous déplacer à l'aide d'une barre de défilement ou modifier une valeur à l'aide d'un bouton toupie, vous pouvez cliquer une fois sur le contrôle pour vous déplacer d'une unité ou maintenir le bouton de la souris enfoncé. Dans ce cas, un premier événement se déclenche. Un court laps de temps s'écoule, puis l'événement s'exécute en boucle à intervalles réguliers jusqu'à ce que vous relâchiez le bouton de la souris. La propriété `Delay` correspond au temps séparant chaque événement exécuté en série. Le laps de temps entre le premier événement et la série qui s'exécute ensuite est égal à cinq fois la valeur de `Delay` ; il évite de déclencher toute une série d'événements sur un seul clic de souris.

Conseil

Ne modifiez pas la valeur par défaut de la propriété `Delay`. Elle correspond au comportement habituel des barres de défilement et des boutons toupies en environnement Windows.

Une valeur trop importante diminue l'efficacité du contrôle. Une valeur trop petite risque de déclencher une série d'événements au moindre clic, alors que l'utilisateur ne souhaite en déclencher qu'un seul.

Max et Min

Les propriétés `Max` et `Min` d'un `ScrollBar` ou d'un `SpinButton` déterminent les valeurs maximale et minimale que peut prendre le contrôle. Elles acceptent une valeur de type `Integer`.

Lorsque vous déplacez le curseur d'un `ScrollBar` ou cliquez sur l'une des flèches d'un `SpinButton`, la propriété `Value` se déplace entre les valeurs `Max` et `Min`. Pour un `ScrollBar`, elle vaut `Min` lorsque le curseur est tout en haut (barre verticale) ou tout à gauche (barre horizontale) et `Max` lorsque le curseur est tout en bas ou à droite du contrôle. Pour un `SpinButton`, elle représente la valeur que sélectionne l'utilisateur, se rapproche de `Min` lorsqu'il clique sur la flèche Haut ou Gauche et vers `Max` lorsqu'il clique sur la flèche Bas ou Droite.

L'unité d'incrémentation de `Value` est déterminée par `SmallChange`, ainsi que par

LargeChange dans le cas d'un ScrollBar. C'est donc la conjugaison de ces trois propriétés qui détermine le nombre de valeurs que peut prendre le contrôle (nombre de positions possibles pour le curseur d'un ScrollBar). Ce nombre est égal à [(Max – Min) / SmallChange] + 1.

Si Min et Max sont respectivement définies à 0 et à 100 et SmallChange à 10, le contrôle peut prendre onze valeurs (0, 10, …, 100). Chaque clic sur la flèche Bas ou Droite incrémentera la valeur de 10, jusqu'à atteindre Max. Au-delà, les clics resteront sans effet. De la même façon, chaque clic sur la flèche Haut ou Gauche ôtera 10 à la valeur, jusqu'à atteindre Min et les clics resteront alors sans effet.

Info

Les valeurs par défaut des propriétés Min et Max d'un ScrollBar sont respectivement 0 et 32 767. La valeur de SmallChange et LargeChange étant définie par défaut à 1, le contrôle ainsi déterminé accepte 32 768 positions.

Les valeurs par défaut des propriétés Min et Max d'un SpinButton sont respectivement 0 et 100. La valeur de SmallChange étant définie par défaut à 1, le contrôle ainsi défini accepte 101 positions.

SmallChange

Cette propriété spécifie la valeur d'incrémentation de la propriété Value d'un ScrollBar ou d'un SpinButton lors des événements SpinDown ou SpinUp. Elle accepte une valeur de type Integer.

Chaque fois que l'utilisateur clique sur la flèche Bas (contrôle vertical) ou Droite (contrôle horizontal), Value est incrémentée de SmallChange, jusqu'à atteindre Max. Les clics restent alors sans effet.

Chaque fois que l'utilisateur clique sur la flèche Haut ou Gauche, Value est décrémentée de SmallChange, jusqu'à atteindre Min. Les clics restent alors sans effet.

Conjuguée aux propriétés Min et Max du contrôle, SmallChange détermine le nombre de valeurs possibles pour le contrôle, correspondant au nombre de positions du curseur sur la barre de défilement dans le cas d'un ScrollBar.

Pour un exemple d'exploitation de ces propriétés, reportez-vous à la section consacrée à l'exploitation des objets SpinButton décrite au chapitre suivant.

LargeChange

Cette propriété spécifie la valeur d'incrémentation de la propriété `Value` d'un `ScrollBar` lorsque l'utilisateur clique dans la barre de défilement, entre le curseur et l'une des flèches de défilement. Elle accepte une valeur de type `Integer` (1 par défaut).

Chaque fois que l'utilisateur clique dans la barre de défilement, entre le curseur et la flèche Bas (contrôle vertical) ou Droite (contrôle horizontal), `Value` est incrémentée de `LargeChange`, jusqu'à atteindre `Max`. Les clics restent alors sans effet.

Chaque fois que l'utilisateur clique dans la barre de défilement, entre le curseur et la flèche Haut ou Gauche, `Value` est décrémentée de `LargeChange`, jusqu'à atteindre `Min`. Les clics restent alors sans effet.

Dans l'environnement Windows, un clic dans une barre de défilement entraîne généralement un déplacement supérieur à un clic sur la flèche (dans une feuille Excel par exemple, le déplacement est respectivement d'un écran et d'une ligne).

Pour un exemple d'utilisation de la propriété `LargeChange`, voyez la section consacrée à l'exploitation des objets `ScrollBar`, au chapitre suivant.

Divers

Accelerator

Cette propriété définit un raccourci clavier pour un contrôle. Elle accepte une valeur de type `String`. Affectez-lui un des caractères formant le texte de sa propriété `Caption`.

La lettre du raccourci est soulignée dans le libellé du contrôle (seulement la première occurrence). Sa casse ne sera considérée que si le libellé du contrôle contient à la fois la lettre majuscule et la lettre minuscule.

Combinée à la touche Alt, la saisie de la touche de raccourci d'un contrôle lui passe le focus. Si l'événement par défaut du contrôle est `Click` (pour un bouton de commande ou une case à cocher, par exemple), alors il s'exécute.

La figure 13-16 présente une feuille dont tous les contrôles ont été affectés à un raccourci clavier.

Lorsqu'une touche de raccourci est affectée à un `Label`, elle entraîne le passage du focus au contrôle suivant dans l'ordre de tabulation. Cette fonction est particulièrement intéressante pour activer un `TextBox`.

Attention

Veillez à ne pas attribuer un même raccourci clavier à des contrôles différents sur une feuille. Sinon, cela entraîne le passage du focus au premier suivant le contrôle actif, dans l'ordre de tabulation.

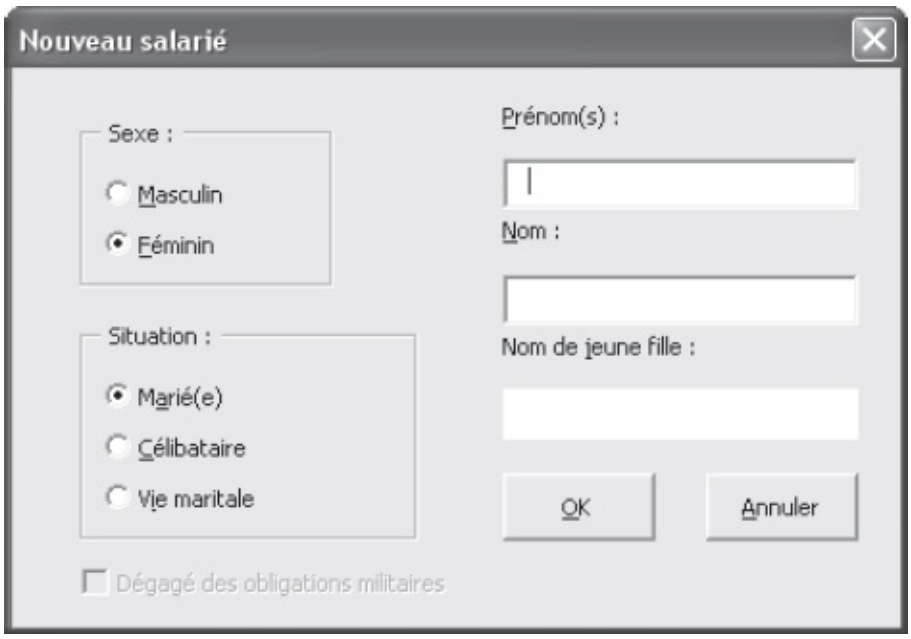


Figure 13-16 – *N'hésitez pas à affecter des touches de raccourci aux contrôles d'une feuille afin d'en optimiser l'utilisation.*

GroupName

Cette propriété permet d'associer des `OptionButton`, afin que l'utilisateur ne puisse valider que l'un d'eux. Elle accepte une valeur de type `String`. Pour associer des boutons d'options, affectez la même chaîne de caractères à leurs propriétés `GroupName` respectives.

Astuce

Notez qu'il existe deux moyens d'associer des boutons d'options sur une feuille : les placer sur un

même cadre (`Frame`) ou affecter une même valeur à leurs propriétés `GroupName` respectives. Cette seconde possibilité prend moins de place et offre plus de liberté puisque les boutons peuvent être placés n'importe où sur la feuille. L'utilisation d'un cadre est cependant souvent ***visuellement*** plus efficace, surtout lorsque la feuille propose plusieurs groupes.

Astuce

Pour définir à l'identique la propriété `GroupName` de plusieurs boutons d'options, sélectionnez-les sur la feuille, ouvrez la fenêtre Propriétés (touche F4) et affectez la valeur de votre choix à `GroupName`. Tous les contrôles sélectionnés sont affectés par cette modification.

HelpContextID

Cette propriété sert à spécifier une rubrique d'aide pour le contrôle. Elle accepte une valeur de type `Long` correspondant à l'identificateur de contexte de l'une des rubriques du fichier d'aide. Cet ouvrage n'aborde pas ce sujet.

MouseIcon

Cette propriété spécifie une *icône souris* personnalisée pour le contrôle, c'est-à-dire l'apparence que prend le pointeur lorsqu'il est placé au-dessus du contrôle. Pour que cette propriété ait un effet, `MousePointer` doit être définie à `fmMousePointerCustom`. À partir de la fenêtre Propriétés :

1. Cliquez sur le bouton Parcourir de la propriété `MouseIcon`. La fenêtre Charger une image s'affiche.
2. Sélectionnez un fichier de format valide (l'extension la plus courante est .ico), puis cliquez sur Ouvrir.

Vous pouvez aussi écrire du code affectant une icône personnalisée à un contrôle en cours d'exécution d'un programme. Faites pour cela appel à la fonction `LoadPicture`, selon la syntaxe suivante :

```
Contrôle.MouseIcon = LoadPicture(NomIcône)
```

Figure 13-17 – *La propriété MouseIcon affecte des icônes personnalisées aux contrôles.*

Attention

Pour affecter un curseur personnalisé à un contrôle, vous devez également affecter la valeur `fmMousePointerCustom` à sa propriété `MousePointer` – que le curseur soit affecté au contrôle lors de la phase de conception de la feuille ou lors de l'exécution du code à l'aide de la fonction `LoadPicture`.

MousePointer

Cette propriété détermine l'apparence du pointeur de souris du contrôle. Elle accepte pour valeur l'une des dix-sept constantes `fmMousePointer`. Les seize premières correspondent à des pointeurs prédéfinis et la dix-septième, `fmMousePointerCustom`, permet de personnaliser l'apparence (voir section précédente).

Pour visualiser les différents pointeurs disponibles, placez sur une feuille un `ComboBox` et un `CommandButton` puis définissez leurs propriétés comme indiqué dans le tableau suivant :

Propriété	Valeur
Feuille	
`Name`	`fmTestPointeur`
Contrôle *`ComboBox`*	
`Name`	`cboConstantes`
`Locked`	`True`
Contrôle *`CommandButton`*	
`Name`	`cmdQuitter`
`Value`	Quitter

Placez ensuite le code suivant dans la section Déclarations de la feuille :

```
Private Sub UserForm_Initialize()
  Dim ValConstante
  For ValConstante = 1 To 16
    cboConstantes.AddItem (ValConstante & "e constante")
  Next ValConstante
End Sub


Private Sub cboConstantes_Click()
  cmdQuitter.MousePointer = cboConstantes.ListIndex
End Sub


Private Sub cmdQuitter_Click()
  fmTestPointeur.Hide
End Sub
```

Sélectionnez ensuite la feuille dans Visual Basic Editor, puis cliquez sur le bouton Exécuter de la barre d'outils Standard. La feuille s'affiche. Sélectionnez une à une les différentes options de la liste déroulante et placez le pointeur sur le bouton pour chaque sélection. Cliquez sur le bouton Quitter pour fermer le programme et retourner à Visual Basic Editor.

Info

Lorsque vous modifiez la propriété `MousePointer` d'un objet `UserForm` (feuille), le pointeur de souris déterminé affecte toutes les feuilles, quel que soit le contrôle au-dessus duquel il se trouve. Pour le vérifier, remplacez l'instruction suivante :

```
cmdQuitter.MousePointer = cboConstantes.ListIndex
```

par :

```
fmTestPointeur.MousePointer = cboConstantes.ListIndex
```

Figure 13-18 – *Vous pouvez modifier l'apparence d'un pointeur à tout moment de l'exécution du programme.*

TabIndex

Cette propriété détermine la position du contrôle dans l'ordre de tabulation de la feuille. Elle accepte une valeur de type `Integer`, comprise entre 0 et le nombre de contrôles de la feuille moins 1.

> **Info**
> Pour un rappel des notions de focus et d'ordre de tabulation, reportez-vous à la section décrivant la propriété `AutoTab`, plus haut dans ce chapitre.

Lorsque l'utilisateur tape sur la touche Tab, le focus passe au contrôle dont la propriété `TabIndex` vaut celle du contrôle actif plus 1. S'il utilise la combinaison Maj+Tab, le focus passe au contrôle dont la propriété `TabIndex` vaut celle du contrôle actif moins 1.

Si la propriété `Enabled` du contrôle suivant est définie à `False`, il est ignoré et le focus passe au contrôle d'après, dans l'ordre de tabulation de la feuille. Si vous souhaitez qu'un contrôle soit ignoré dans l'ordre de tabulation tout en restant accessible à l'utilisateur, définissez sa propriété `TabStop` à `False`.

Pour définir l'ordre de tabulation d'une feuille en phase de conception, il est plus simple d'utiliser la boîte de dialogue Ordre de tabulation que de définir une à une les `TabIndex` des contrôles :

1. Cliquez-droit sur la feuille (pas sur un contrôle) et choisissez la commande Ordre de tabulation (voir figure 13-19). Les contrôles de la feuille sont répertoriés par leur nom (`Name`) et classés selon leur position dans l'ordre de tabulation (celui dont la propriété `TabIndex` est égale à 0 se trouve tout en haut de la liste).
2. Sélectionnez un contrôle et cliquez sur le bouton Vers le haut ou Vers le bas pour le déplacer dans l'ordre de tabulation. À chaque mouvement, sa propriété `TabIndex` et celle du contrôle dont il prend la place sont interverties.
3. Vous pouvez déplacer simultanément plusieurs contrôles tout en conservant leurs positions relatives. Pour sélectionner des contrôles contigus (resp. non contigus), utilisez la touche Maj (resp. Ctrl).

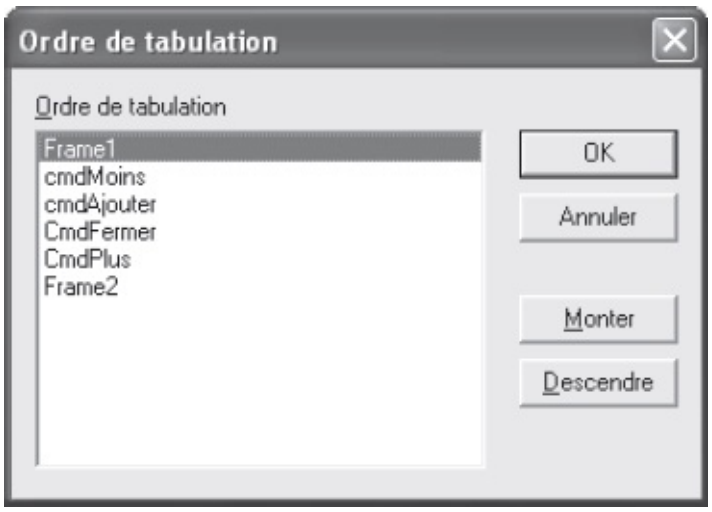


Figure 13-19 – *La boîte de dialogue Ordre de tabulation.*

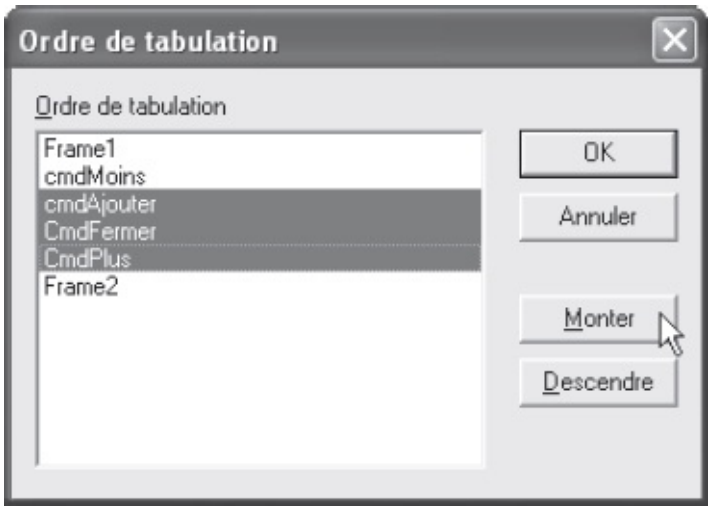


Figure 13-20 – *Déplacez simultanément les contrôles dont vous souhaitez modifier la position dans l'ordre de tabulation, tout en conservant leurs positions relatives.*

TabStop

Cette propriété détermine si un contrôle peut ou non recevoir le focus lorsque l'utilisateur tape sur la touche Tab ou sur la combinaison Maj+Tab. Elle accepte une valeur de type `Boolean` :

- **`True`** (par défaut). Si sa position dans l'ordre de tabulation le justifie, le contrôle reçoit le focus lorsque la touche Tab ou la combinaison Maj+Tab est activée.
- **`False`**. Le focus ne peut être passé au contrôle par la touche Tab. Bien qu'il

conserve sa propriété `TabIndex`, il est ignoré s'il est le suivant (touche Tab seule) ou le précédent (Maj+Tab) dans l'ordre de tabulation, et le focus est passé à celui d'après.

Tag

Cette propriété stocke des informations supplémentaires sur le contrôle sous la forme d'une chaîne de caractères. Il peut s'agir d'une description du contrôle, affichée à l'utilisateur si nécessaire.

Emplacement

Height et Width

Ces propriétés, de type `Single`, déterminent respectivement la hauteur et la largeur en points d'un contrôle.

Vous n'avez *a priori* pas à vous en préoccuper, puisqu'elles sont automatiquement mises à jour lorsque vous redimensionnez le contrôle dans Visual Basic Editor. Préférez donc les méthodes de dimensionnement présentées au chapitre précédent.

Left et Top

Ces propriétés, de type `Single`, déterminent la distance en points du contrôle, respectivement par rapport au bord gauche et au bord supérieur de l'objet qui le contient.

Vous n'avez *a priori* pas à vous en préoccuper, puisqu'elles sont automatiquement mises à jour lorsque vous déplacez le contrôle dans Visual Basic Editor. Préférez donc les méthodes de mise en forme présentées au chapitre précédent.

Dans le cas d'une feuille, les propriétés `Left` et `Top` représentent la distance en points de la feuille par rapport aux bords gauche et supérieur de l'écran lors de son affichage. Elles ne sont considérées que si la propriété `StartUpPosition` de la feuille est définie à 0 (manuel).

StartUpPosition

Cette propriété détermine la position de la feuille sur l'écran. Elle accepte l'une des valeurs suivantes :

- `0 - Manual`. La position de la feuille sur l'écran est déterminée par ses propriétés `Left` et `Top`.
- `1 - CenterOwner` (par défaut). La feuille est centrée sur la fenêtre de l'application hôte.
- `2 - CenterScreen`. La feuille est centrée sur l'écran, quelles que soient la position et la taille de la fenêtre de l'application hôte.
- `3 - Windows Default`. La feuille s'affiche dans l'angle supérieur gauche de l'écran. Utilisez cette valeur lorsque vous souhaitez que la feuille affichée masque le moins possible le document actif.

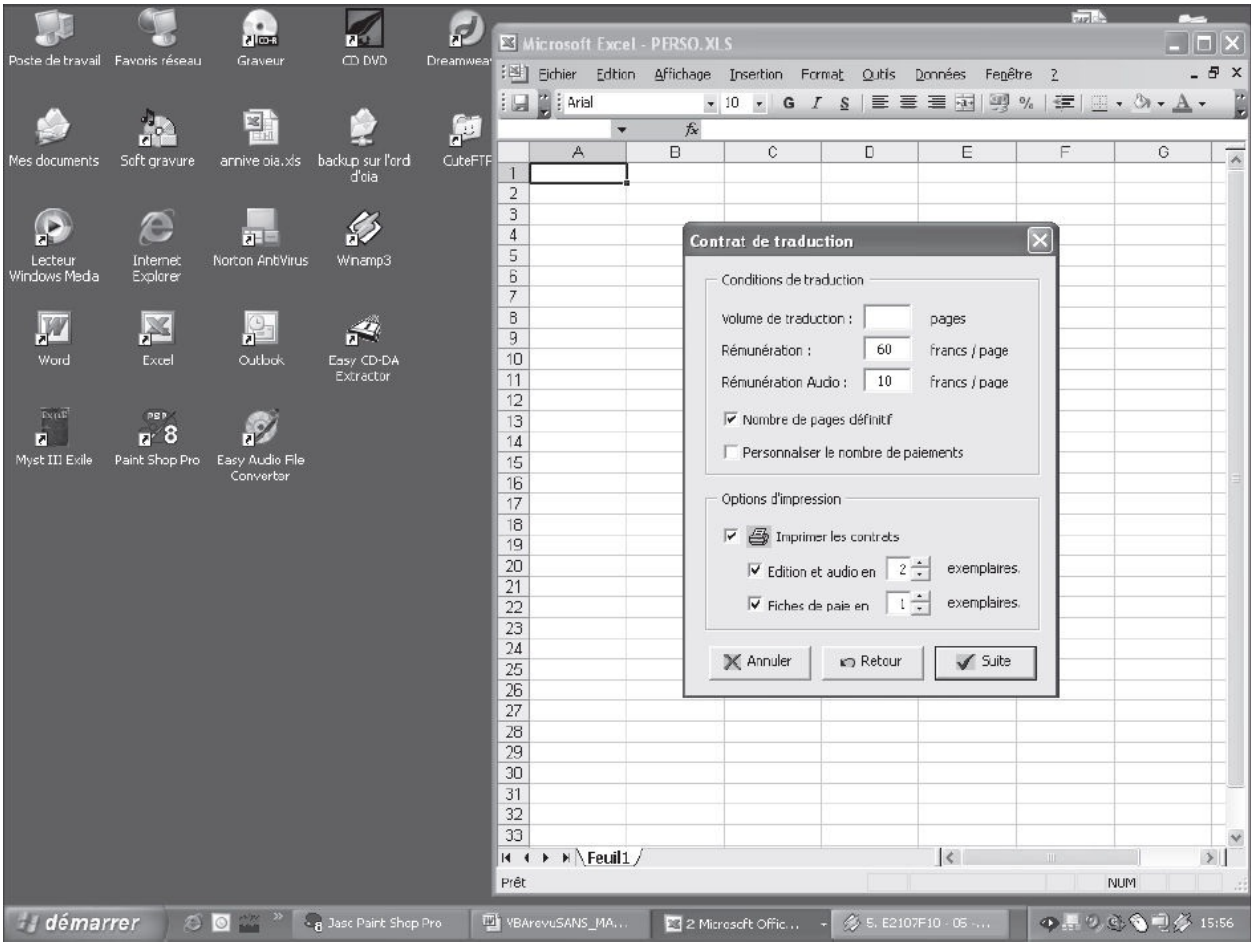
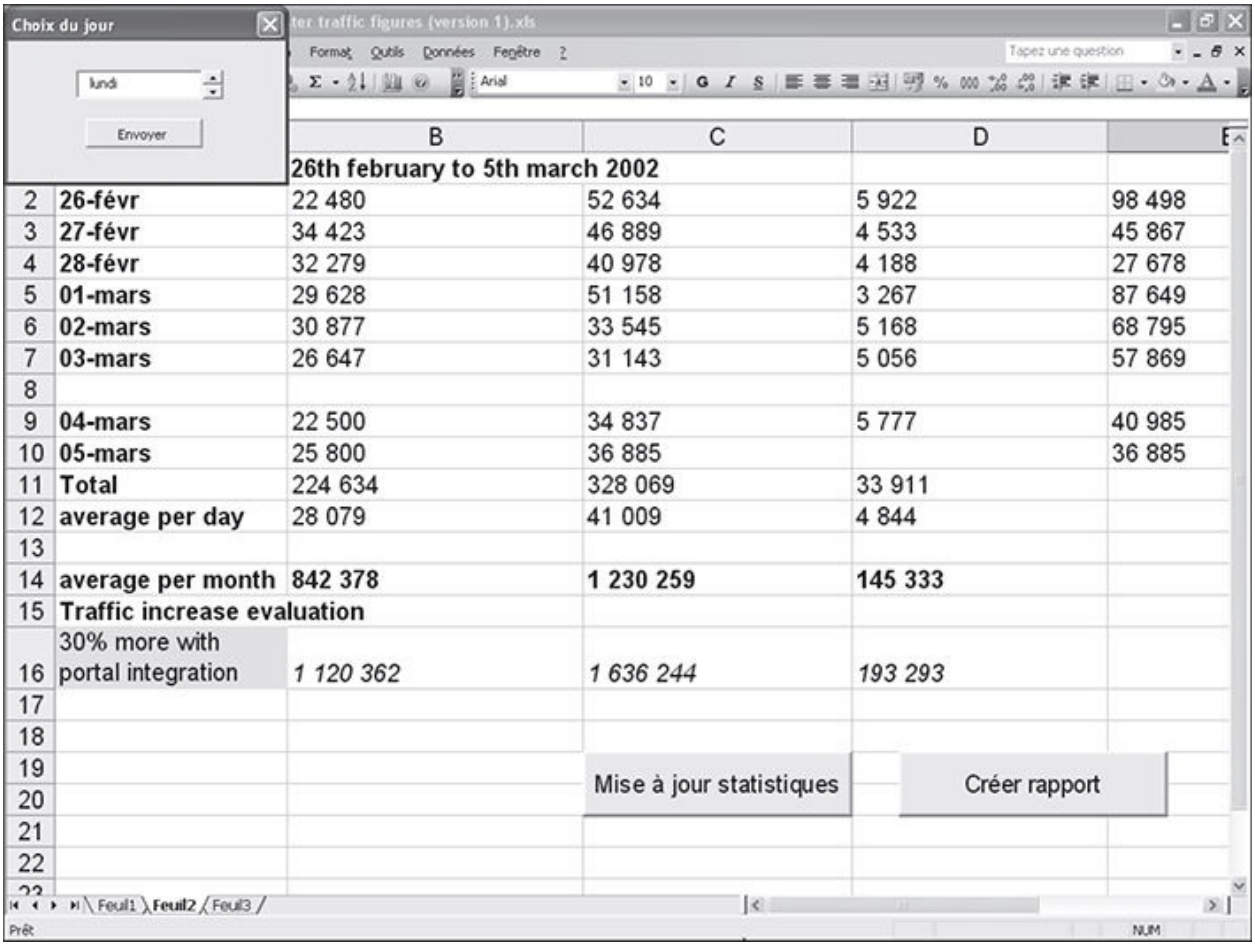


Figure 13-21 – *Vous pouvez choisir de toujours afficher la feuille au centre de la fenêtre de l'application hôte, quelles que soient la taille et la position de cette dernière.*



	A	B	C	D	E
		26th february to 5th march 2002			
2	26-févr	22 480	52 634	5 922	98 498
3	27-févr	34 423	46 889	4 533	45 867
4	28-févr	32 279	40 978	4 188	27 678
5	01-mars	29 628	51 158	3 267	87 649
6	02-mars	30 877	33 545	5 168	68 795
7	03-mars	26 647	31 143	5 056	57 869
8					
9	04-mars	22 500	34 837	5 777	40 985
10	05-mars	25 800	36 885		36 885
11	Total	224 634	328 069	33 911	
12	average per day	28 079	41 009	4 844	
13					
14	average per month	842 378	1 230 259	145 333	
15	Traffic increase evaluation				
16	30% more with portal integration	*1 120 362*	*1 636 244*	*193 293*	
17					
18					
19					
20					
21					
22					

Figure 13-22 – *L'affichage de la feuille dans l'angle supérieur gauche de l'écran évite que des données importantes du document ne soient masquées.*

Image

Picture

Cette propriété affecte une image à un contrôle ou à une feuille :

1. Affichez la fenêtre Propriétés de l'objet et cliquez sur le bouton Parcourir (…) de `Picture`. La boîte de dialogue Charger une image s'affiche (voir figure 13-23).

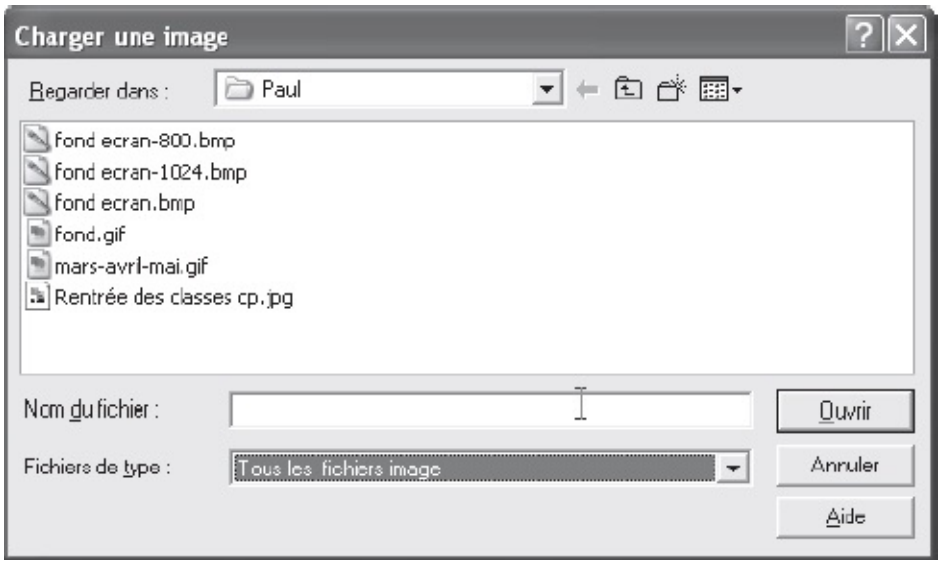


Figure 13-23 – *Sélectionnez le fichier image que vous souhaitez affecter au contrôle.*

2. Sélectionnez un fichier de format valide qui sera utilisé comme image pour le contrôle ou la feuille (.bmp et .ico sont des formats valides), puis cliquez sur Ouvrir.

Vous pouvez aussi écrire du code affectant une image à un objet en cours d'exécution d'un programme, grâce à la fonction `LoadPicture`, selon la syntaxe suivante :

```
Objet.Picture = LoadPicture(NomImage)
```

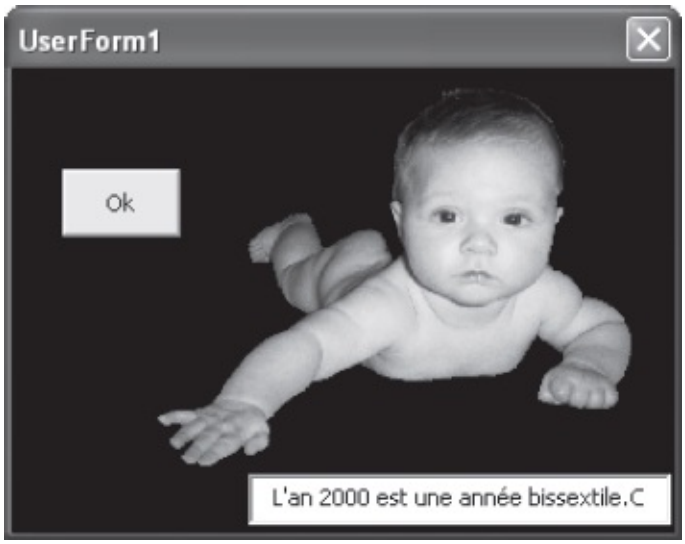


Figure 13-24 – *Vous pouvez donner un peu de fantaisie à vos feuilles en leur affectant des images.*

Pour supprimer une image d'un contrôle, sélectionnez la valeur de `Picture` dans la fenêtre Propriétés et appuyez sur la touche Suppr.

PictureAlignment

Cette propriété détermine l'emplacement d'une image (propriété `Picture`) sur un contrôle ou sur une feuille. Elle accepte pour valeur l'une des constantes `fmPictureAlignment` :

- **`fmPictureAlignmentTopLeft`**. L'image est placée dans l'angle supérieur gauche de l'objet.
- **`fmPictureAlignmentTopRight`**. L'image est placée dans l'angle supérieur droit de l'objet.
- **`fmPictureAlignmentCenter`** (par défaut). L'image est centrée sur l'objet.
- **`fmPictureAlignmentBottomLeft`**. L'image est placée dans l'angle inférieur gauche de l'objet.
- **`fmPictureAlignmentBottomRight`**. L'image est placée dans l'angle inférieur droit de l'objet.

Pour les `CommandButton`, `PictureAlignment` est remplacée par `PicturePosition`. Utilisez cette propriété pour affecter une image à un bouton de commande sans en masquer le libellé. Par exemple, une image a été affectée à la propriété `Picture` des trois `CommandButton` de la figure 13-25. `PicturePosition` a été définie sur `fmPicturePositionLeftCenter`. Notez le contrôle (et non la propriété) `Picture` représentant une imprimante.

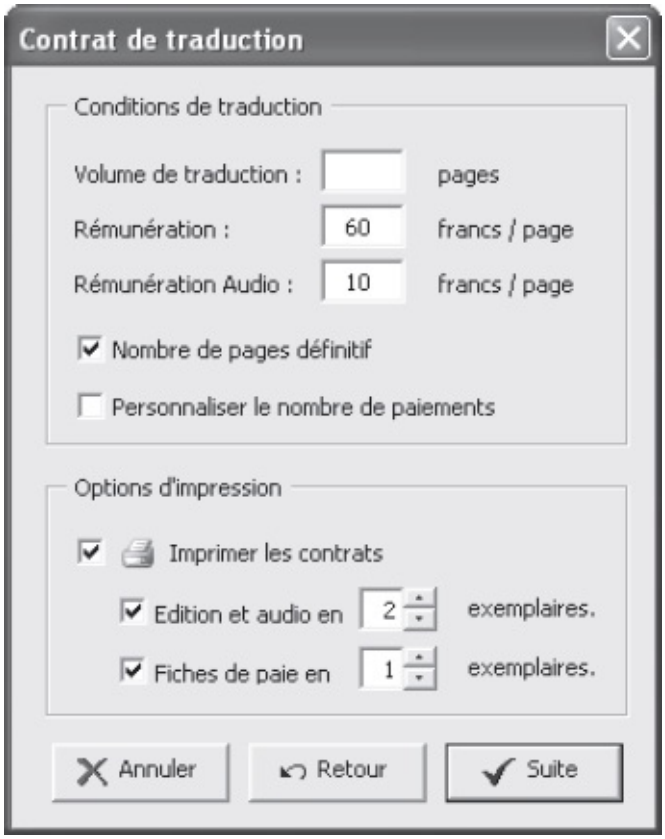


Figure 13-25 – *L'affectation d'une image à la propriété Picture des boutons de commande de vos feuilles leur donnera un peu de gaieté.*

Astuce

Réalisez une capture d'écran d'une fenêtre dont vous souhaitez exploiter les icônes. Détourez ces dernières dans un logiciel de dessin tel que Paint Shop Pro et enregistrez-les au format BMP. Vous pourrez ensuite les exploiter dans vos feuilles VBA comme contrôles `Picture` ou comme propriétés `Picture` de `CommandButton`.

PicturePosition

Cette propriété détermine la position de l'image d'un `CommandButton` par rapport à sa légende. Elle accepte pour valeur l'une des treize constantes `fmPicturePosition`. Pour plus d'informations, reportez-vous à l'Aide Visual Basic.

PictureSizeMode

Cette propriété détermine de quelle façon l'image d'un contrôle ou d'une feuille s'affiche si sa taille diffère de celle de l'objet conteneur. Elle n'est pas

gérée par les `CommandButton`. Elle accepte pour valeur l'une des constantes `fmPictureSizeMode` :

- **`fmPictureSizeModeClip`** (par défaut). Si la taille de l'image est supérieure à celle du contrôle, seule la partie tenant dans le contrôle s'affiche. La partie rognée variera en fonction de `PictureAlignment`.
- **`fmPictureSizeModeStretch`**. L'image est redimensionnée à la même taille que l'objet conteneur. Si le rapport homothétique n'est pas équilibré, l'image sera déformée.
- **`fmPictureSizeModeZoom`**. L'image est redimensionnée proportionnellement. Si le rapport homothétique n'est pas équilibré, l'image occupera toute la largeur de l'objet conteneur, mais pas toute la hauteur, ou inversement.

PictureTiling

Cette propriété détermine si une image est ou non affichée en mosaïque sur une feuille ou un contrôle. Elle accepte une valeur de type `Boolean` :

- **`True`**. Si l'image est plus petite que l'objet conteneur, elle s'affiche en mosaïque sur la page (voir figure 13-26). Pour que l'affichage en mosaïque s'effectue correctement, la propriété `PictureSizeMode` doit être définie à `fmPictureSizeModeClip`.
- **`False`** (par défaut). L'image n'est pas affichée en mosaïque.

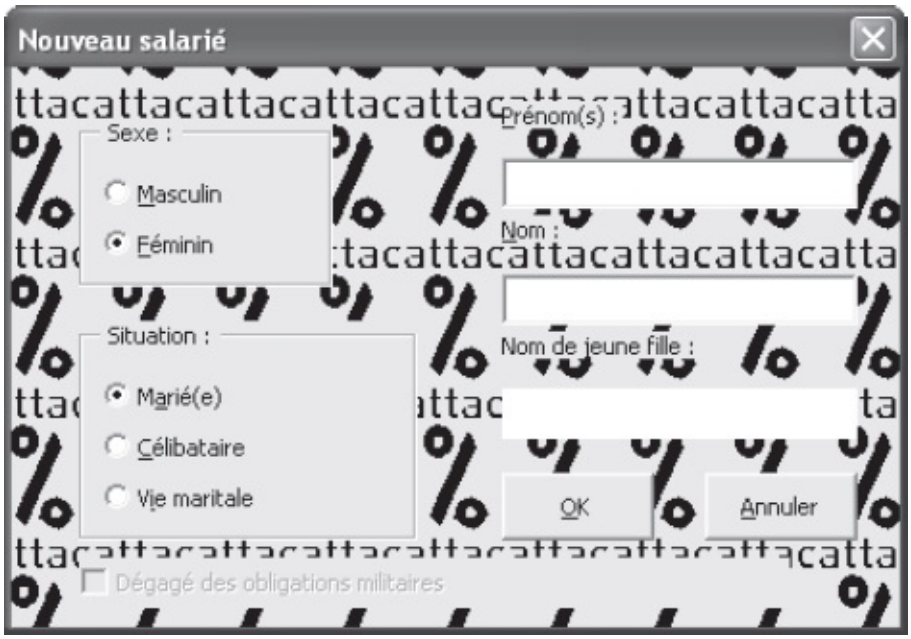


Figure 13-26 – *Une image affichée en mosaïque sur une feuille.*

Info

Une image affichée en mosaïque sur une feuille ou un contrôle risque d'être rognée sur le bord supérieur ou inférieur de l'objet, ainsi que sur son bord gauche ou droit. Il est en effet peu probable que la hauteur et la largeur de l'objet conteneur soient des multiples de celles de l'image. Les bords rognés dépendront de l'alignement de l'image sur l'objet (`PictureAlignment`).

Police

Font

Cette propriété détermine la police de caractères affectée à l'affichage du contenu d'un contrôle (Tahoma Regular corps 8, par défaut). Pour la modifier, cliquez sur le bouton … dans le volet droit de la fenêtre Propriétés du contrôle. Dans la boîte de dialogue qui s'affiche, déterminez la police de caractères et les attributs voulus, puis cliquez sur OK.

14

Maîtriser le comportement des contrôles

Ce chapitre vous propose de découvrir comment rendre les interfaces utilisateur vraiment fonctionnelles en leur ajoutant de l'interactivité grâce aux procédures événementielles. Il vous présente également les techniques les plus courantes pour tirer parti des propriétés de chaque contrôle.

Créer des procédures événementielles

Les contrôles placés sur une feuille sont réceptifs aux événements utilisateur qui les affectent. Vous pouvez ainsi créer des procédures dites événementielles, qui se déclencheront lorsque l'événement correspondant (un clic de souris, par exemple) sera repéré. Ces procédures ont des fonctions très variées, telles que vérifier la validité d'une information, modifier l'apparence de la feuille, ouvrir une autre feuille, fermer la feuille et passer les valeurs qu'elle contient à une procédure du module de code, etc.

Créer une procédure

Il existe plusieurs façons de créer des procédures événementielles. Quelle que soit la méthode choisie, elles répondent toujours à une même syntaxe :

```
Private Sub Contrôle_Evénement()
   Instructions
End Sub
```

L'instruction de déclaration est toujours précédée de `Private`, car une procédure événementielle est par définition privée. Elle ne s'exécute que lorsque l'événement est repéré et ne peut être appelée par une instruction située dans une procédure du projet.

`Contrôle` est le nom (propriété `Name`) du contrôle auquel est attachée la procédure. *`Evénement`* déclenchera la procédure lorsqu'il affectera le contrôle ; il peut s'agir d'un clic de souris, d'une frappe de la touche Entrée ou encore d'une

modification de la valeur du contrôle. Le nom du contrôle et celui de l'événement sont toujours séparés par un trait de soulignement.

Info

Dans le cas de procédures événementielles affectées à des feuilles, le nom de l'objet n'apparaît pas dans l'instruction de déclaration. Le mot-clé `UserForm` est dans ce cas employé pour identifier la feuille :

```
Private Sub UserForm_Evénement()
```

Vous pouvez écrire une procédure événementielle directement dans la fenêtre de Code de la feuille, en adoptant l'une des méthodes présentées plus loin dans ce chapitre. Cliquez-droit n'importe où sur la feuille et choisissez la commande Code.

Les instructions de déclaration peuvent cependant être ajoutées automatiquement à partir de la feuille. Pour affecter une procédure événementielle à un contrôle, à partir de l'objet conteneur :

1. Double-cliquez sur le contrôle voulu, ou cliquez-droit et choisissez la commande Code (voir figure 14-1), ou sélectionnez le contrôle puis choisissez la commande Code du menu Affichage.

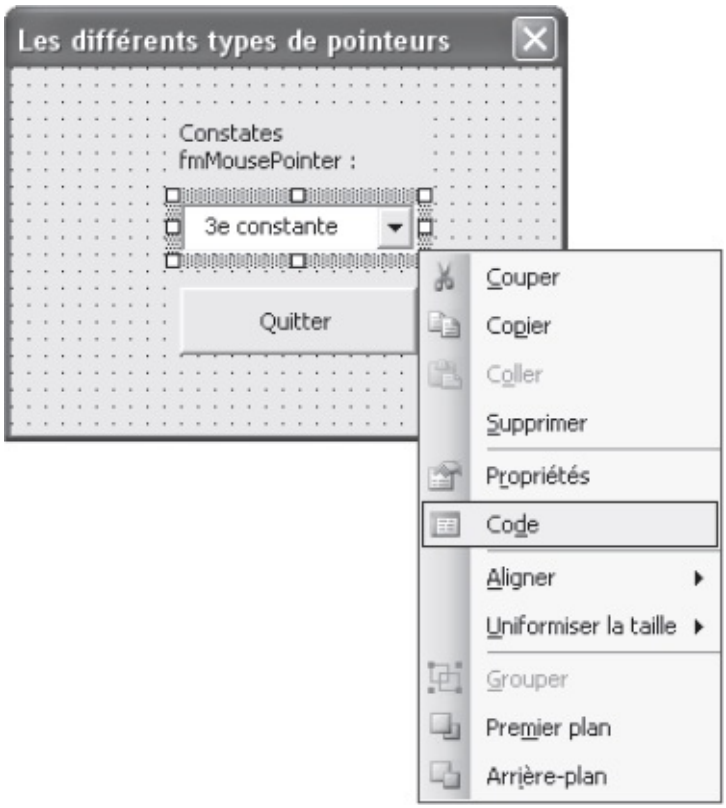


Figure 14-1 – *Ouvrez la fenêtre Code à partir du contrôle auquel vous souhaitez affecter une procédure événementielle.*

2. La fenêtre Code de la feuille s'ouvre. Les instructions de déclaration par défaut de la procédure événementielle du contrôle sont automatiquement insérées et le curseur est placé entre elles (voir figure 14-2).

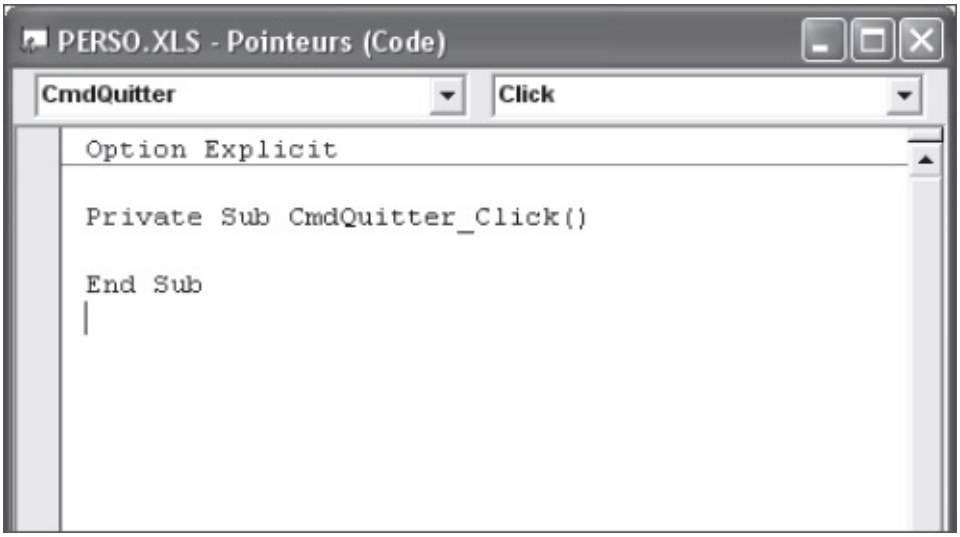


Figure 14-2 – *Les instructions de déclaration de la procédure sont automatiquement ajoutées.*

L'événement affecté à la procédure est l'événement par défaut du contrôle. Il varie d'un contrôle à l'autre : `Click` (clic de souris) pour un `CommandButton`, `Change` (modification de la valeur) pour un `TextBox`...

3. Si l'événement n'est pas celui que vous voulez, modifiez-le au clavier, ou déroulez la liste Procédure pour faire votre choix (voir figure 14-3) parmi tous ceux susceptibles d'affecter le contrôle et supprimez les déclarations précédentes.
4. Saisissez le code de la procédure entre les instructions de déclaration.

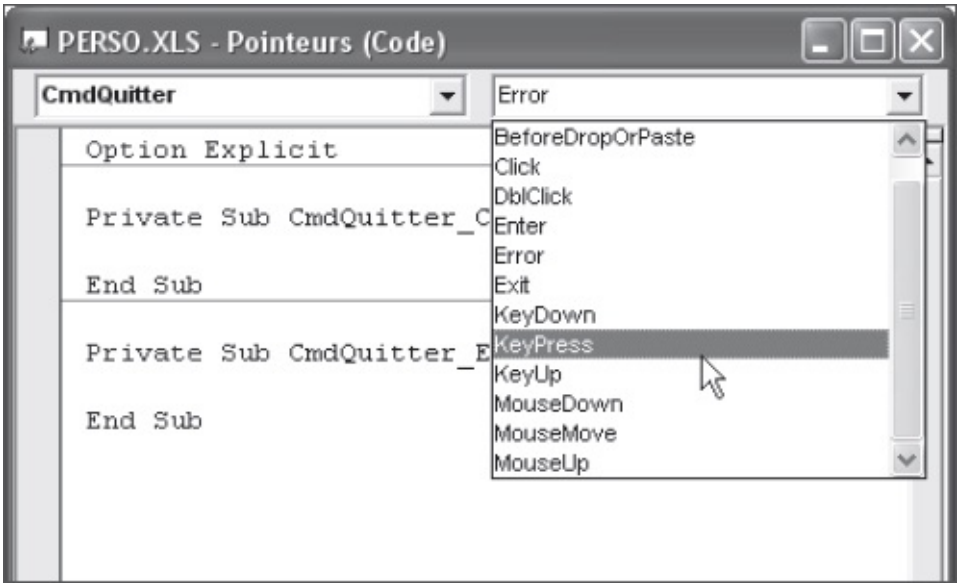


Figure 14-3 – *La liste déroulante Procédure propose tous les événements susceptibles d'être détectés sur le contrôle affiché dans la zone Objet.*

Info

Si la procédure événementielle existe déjà, le curseur est placé sous son instruction de déclaration.

Pour réaliser l'exemple suivant, créez une feuille et placez-y un `Label`, un `TextBox`, quatre `OptionButton` et trois `CommandButton`, en définissant leurs propriétés selon le tableau suivant.

Propriété	Valeur
Feuille	
`Name`	`fmTestEvénements`
`Caption`	Test des procédures événementielles
Contrôle *`Label`*	
`Name`	`lbTexte`
`Value`	Entrez le texte de votre choix
Contrôle *`TextBox`*	
`Name`	`txtTexte`
`Value`	
Contrôles *`OptionButton`*	
`Name`	De `optOption1` à `optOption4`
`Value`	`True` pour l'un des boutons ; `False` pour les autres
`Caption`	De Bouton d'option 1 à Bouton d'option 4
Contrôles *`CommandButton`*	

Name	Respectivement cmdBouton, cmdTexte et cmdQuitter
Caption	Respectivement Bouton actif, Zone de texte et Quitter

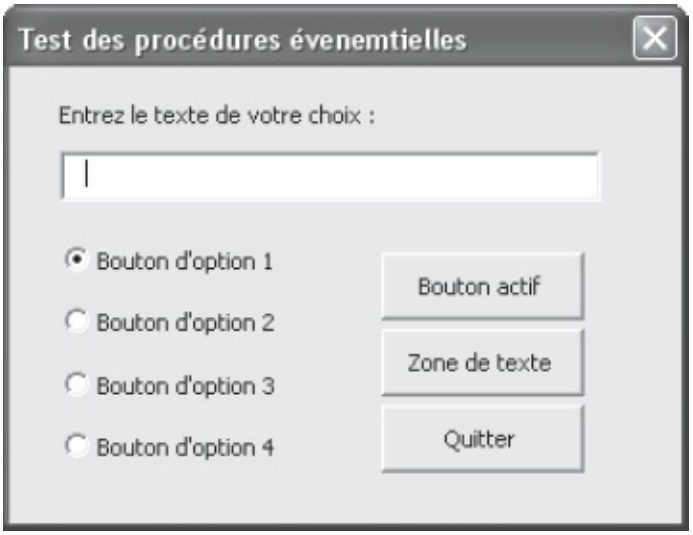


Figure 14-4 – *La feuille Test des procédures événementielles réalisée.*

Enfin, affectez les procédures événementielles suivantes au `TextBox` et aux trois `CommandButton`.

```
Private Sub txtTexte_AfterUpdate()
  MsgBox "Vous avez modifié la valeur de la zone de texte.", _
    vbOKOnly + vbInformation, "Evénement AfterUpdate() détecté"
End Sub


Private Sub cmdBouton_Click()
  Dim BoutonActif As String
  If optOption1.Value=True Then
    BoutonActif = optOption1.Caption
  ElseIf optOption2.Value=True Then
    BoutonActif = optOption2.Caption
  ElseIf optOption3.Value=True Then
    BoutonActif = optOption3.Caption
  Else
    BoutonActif = optOption4.Caption
  End If
  MsgBox "Le bouton d'option sélectionné est le bouton libellé " _
    & BoutonActif, vbOKOnly + vbInformation, "Evénement Click() détecté"
End Sub


Private Sub cmdTexte_Click()
  MsgBox "La valeur de la zone de texte est : " & txtTexte.Value, _
    vbOKOnly + vbInformation, "Evénement Click() détecté"
End Sub


Private Sub cmdQuitter_Click()
  fmTestEvénements.Hide
End Sub
```

Sélectionnez la feuille dans la fenêtre UserForm et cliquez sur le bouton Exécuter de la barre d'outils Standard. La feuille s'affiche à l'écran. Testez son comportement. Lorsque les événements affectés à une procédure événementielle sont détectés, cette dernière s'exécute :

- La modification de la valeur de la zone de texte est interprétée comme l'événement `AfterUpdate` affectant `txtTexte` lors du passage du focus à un autre contrôle. La procédure correspondante est exécutée et la boîte de dialogue de la figure 14-5 s'affiche.
- Un clic sur Bouton actif est détecté comme l'événement `Click` affectant `cmdBouton`. La procédure correspondante est exécutée et la boîte de dialogue de la figure 14-6 s'affiche.
- Un clic sur Zone de texte est détecté comme l'événement `Click` affectant `cmdTexte`. La procédure correspondante est exécutée et la boîte de dialogue de la figure 14-7 s'affiche.
- Un clic sur Quitter est détecté comme l'événement `Click` affectant `cmdQuitter`. La procédure correspondante est exécutée et la fenêtre se ferme.

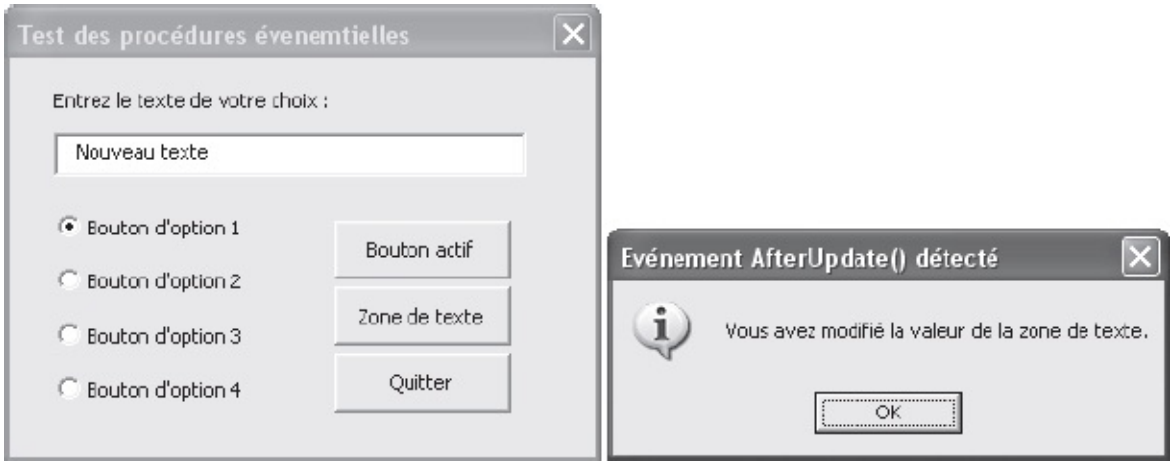


Figure 14-5 – *Déclenchement de la procédure txtTexte_AfterUpdate.*

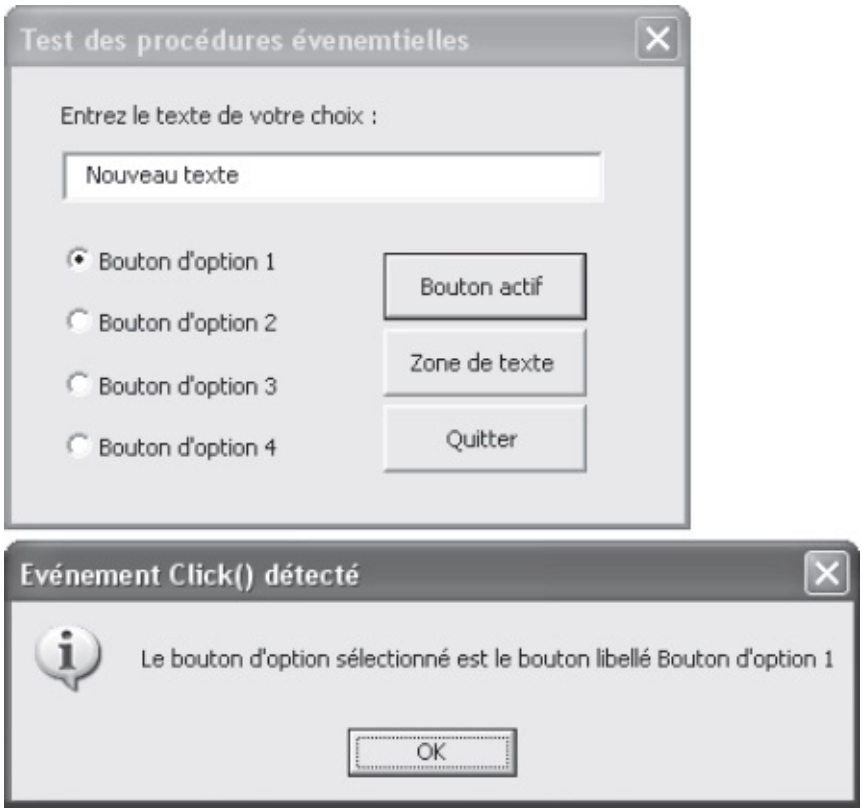


Figure 14-6 – *Déclenchement de la procédure cmdBouton_Click.*

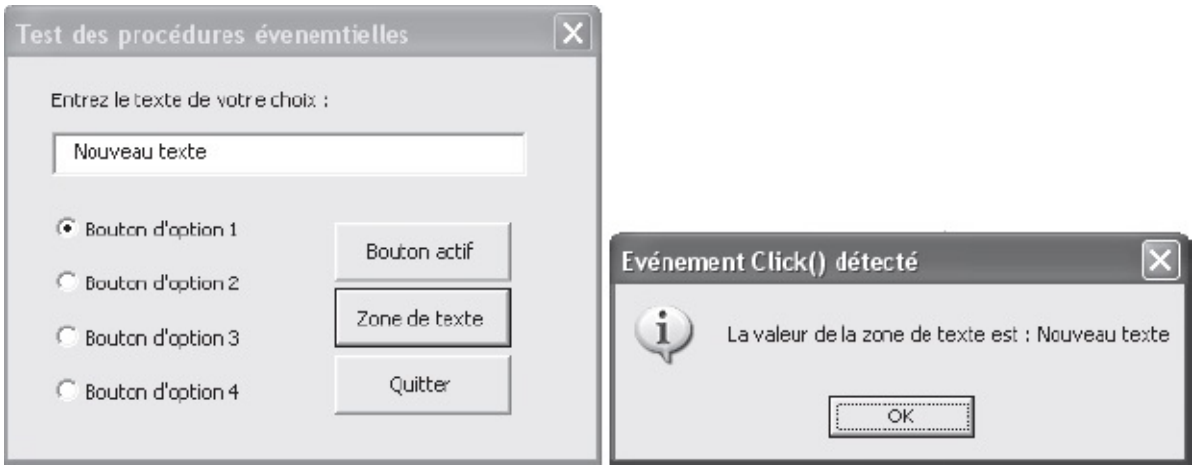


Figure 14-7 – *Déclenchement de la procédure cmdTexte_Click.*

Les événements

Les événements sont nombreux et varient d'un contrôle à l'autre. Ceux qu'un contrôle sait gérer sont intimement liés à sa nature, c'est-à-dire à ses propriétés. Par exemple, les événements `Change`, `AfterUpdate` et `BeforeUpdate` sont

gérés uniquement par les contrôles possédant une propriété `Value` (les `TextBox`, par exemple, mais pas les `CommandButton`), puisqu'ils se déclenchent lors de la modification de cette propriété.

Astuce

Pour accéder à la liste des événements gérés, sélectionnez le contrôle sur une feuille et tapez sur F1 pour ouvrir la rubrique d'aide qui lui est associée.

Cette section présente les événements gérés par les contrôles VBA. Certains, tels que `Click`, `Change` ou `Initialize`, sont essentiels. Ils apparaissent en gras dans la liste. Il en est d'autres que vous n'utiliserez que très rarement, voire jamais.

- **`AfterUpdate`** est détecté lorsque la valeur du contrôle est modifiée, au moment du passage du focus à un autre contrôle. Cet événement survient après `BeforeUpdate`.
- **`BeforeDragOver`** est détecté lors d'un glisser-déplacer.
- **`BeforeDropOrPaste`** est détecté lorsque des données sont déposées ou collées sur un objet. L'événement survient avant que l'action ne soit validée.
- **`BeforeUpdate`** est détecté lorsque la valeur du contrôle est modifiée, au moment du passage du focus à un autre contrôle. Cet événement survient avant que la modification des données ne soit validée – et donc avant `AfterUpdate`. Cela permet de refuser la mise à jour effectuée. `BeforeUpdate` répond à la syntaxe suivante :

```
Private Sub Contrôle_BeforeUpdate(ByVal Annulation As MSForms.ReturnBoolean)
```

Pour refuser la mise à jour, placez l'instruction `Annulation = True` dans la procédure. L'utilisateur ne pourra alors pas passer le focus à un autre contrôle tant qu'une valeur valide n'aura pas été affectée au contrôle.

Dans l'exemple suivant, si l'utilisateur n'entre pas un nombre compris entre 50 et 250 dans la zone de texte `txtTaille`, un message l'avertit que la valeur n'est pas valide et le focus ne peut être passé à un autre contrôle tant que ce n'est pas corrigé.

```
 1: Private Sub txtValeur_BeforeUpdate(ByVal Annulation _
    As MSForms.ReturnBoolean)
 2:   If IsNumeric(txtValeur.Value)=False Then
 3:     MsgBox "Vous devez indiquer une valeur numérique"
 4:     Annulation = True
 5:   Else
 6:     If Not (txtValeur.Value>=50 And txtValeur.Value<=250) Then
 7:       MsgBox "Valeur non valide"
 8:       Annulation = True
 9:     End If
10:   End If
```

```
11: End Sub
```

L'instruction `If…End If` principale (des lignes 2 à 10) vérifie si la valeur entrée est un nombre. Si tel n'est pas le cas (ligne 2), un message s'affiche (ligne 3) et la modification est refusée (ligne 4), interdisant à l'utilisateur de passer le focus à un autre contrôle. Si la valeur entrée est un nombre (ligne 5), une instruction `If…End If` imbriquée (des lignes 6 à 9) vérifie qu'il est compris entre 50 et 250. Si tel n'est pas le cas, un message s'affiche (ligne 7) et la modification est refusée (ligne 8)

- **`Change`** est détecté lors de la modification de la valeur (`Value`) d'un contrôle. La modification provient soit de l'utilisateur, soit d'une instruction du code. Contrairement à `AfterUpdate` et `BeforeUpdate` qui sont déclenchés lors du passage du focus à un autre objet, `Change` est détecté à chaque modification de la valeur. Ainsi, si vous affectez une procédure d'événement `Change` à un `TextBox`, elle s'exécutera chaque fois que l'utilisateur entrera ou supprimera un caractère dans la zone de texte (s'il saisit le mot « Bonjour » dans la zone de texte, la procédure se déclenchera sept fois).

  Il est fréquent d'utiliser une zone de texte pour afficher l'élément sélectionné par l'utilisateur dans une liste. La procédure événementielle suivante met à jour la zone de texte `txtSélListe` chaque fois que l'utilisateur modifie la sélection dans la liste `ZoneDeListe` :

```
Private Sub ZoneDeListe_Change()
  txtSélListe.Value = ZoneDeListe.Value
End Sub
```

- **`Click`** est détecté lorsque l'utilisateur clique sur un contrôle, mais aussi quand l'équivalent clavier d'un clic de souris est tapé. C'est le cas si l'utilisateur frappe la touche Entrée alors qu'un bouton de commande a le focus, ou la barre d'espace alors qu'une case à cocher a le focus. Lorsque vous cliquez sur un contrôle, trois événements sont successivement détectés : `MouseDown`, `MouseUp`, puis `Click`.
- **`dblClick`** est détecté lorsque l'utilisateur double-clique sur un contrôle. Lors d'un double-clic, quatre événements sont donc successivement détectés : `MouseDown`, `MouseUp`, `Click`, puis `dblClick`.
- **`DropButtonClick`** est détecté chaque fois que l'utilisateur déroule ou ferme une liste déroulante modifiable.
- **`Enter`** est détecté juste avant qu'un contrôle reçoive le focus. Il est toujours lié à un événement `Exit` affectant le contrôle qui avait le focus précédemment.
- **`Exit`** est détecté juste avant qu'un contrôle ne perde le focus. Il répond à la

syntaxe suivante :

```
Private Sub Contrôle_Exit(ByVal Annulation As MSForms.ReturnBoolean)
```

Pour annuler le passage du focus, affectez la valeur `True` à l'argument `Annulation` dans une instruction de la procédure événementielle (voir l'exemple donné pour `BeforeUpdate`).

- **`Initialize`** est détecté lorsqu'une feuille est chargée. La procédure affectée à cet événement s'exécute avant l'affichage de la feuille et s'utilise selon la syntaxe suivante :

```
Private Sub UserForm_Initialize()
   Instructions
End Sub
```

Notez que, dans le cas de la procédure `Initialize` d'une feuille, le nom de cette dernière n'est pas utilisé pour spécifier l'objet ; on utilise toujours le mot-clé `UserForm`.

C'est une procédure absolument essentielle de la programmation VBA. Elle permet en effet d'effectuer des réglages de la feuille avant son affichage. Vous l'utiliserez, par exemple, pour affecter une liste d'éléments à un `ListBox` ou un `ComboBox` de la feuille. Reportez-vous à la section « ComboBox », plus loin dans ce chapitre.

Attention

Lorsque vous utilisez la méthode `Hide` pour masquer une feuille précédemment affichée à l'aide de `Show`, les ressources mémoire qu'elle occupe ne sont pas libérées. Si, par la suite, vous affichez de nouveau la feuille, l'événement `Initialize` ne sera donc pas reconnu et ses différents contrôles auront les mêmes valeurs que lorsqu'elle a été masquée. Pour libérer les ressources mémoire d'une feuille UserForm, vous devez lui appliquer la méthode `Unload`, selon la syntaxe suivante :

```
Unload NomFeuille
```

La méthode `Load` charge une feuille en déclenchant la procédure événementielle `Initialize`, sans pour autant l'afficher. Notez que, pour manipuler une feuille par programmation, il faut qu'elle soit chargée (à l'aide de `Show` ou de `Load`).

- **`KeyDown`** est détecté lorsqu'une touche du clavier est enfoncée. Il répond à la syntaxe suivante :

```
Private Sub Contrôle_KeyDown(ByVal CodeTouche As MSForms.ReturnInteger,
ByVal EtatMaj As Integer)
```

`CodeTouche` renvoie le code de la touche frappée et `EtatMaj` renvoie l'état des touches Maj, Ctrl et Alt (0 si aucune n'est enfoncée, 1 si Maj est enfoncée, 2 si c'est Ctrl, 4 si c'est Alt, ces valeurs s'additionnant si plusieurs de ces touches sont enfoncées).

Lorsque l'utilisateur enfonce une touche, `KeyDown` et `KeyPress` sont successivement détectés. S'il maintient la touche enfoncée, ces événements sont à nouveau détectés en série, le processus reflétant la périodicité d'insertion du caractère.

- **`KeyPress`** est détecté lorsque l'utilisateur appuie sur une touche ANSI. Il intervient immédiatement après `KeyDown`. Si la touche reste enfoncée, `KeyPress` est détecté en série, à chaque insertion d'un caractère. Il répond à la syntaxe suivante :

```
Private Sub Contrôle_Keypress(ByVal codeANSI As MSForms.ReturnInteger)
```

- **`KeyUp`** est détecté lorsqu'une touche est relâchée. Il répond à la syntaxe suivante :

```
Private Sub Contrôle_KeyUp(ByVal CodeTouche As MSForms.ReturnInteger,
ByVal EtatMaj As Integer)
```

Les valeurs affectées aux variables `CodeTouche` et `EtatMaj` sont les mêmes que pour `KeyDown`.

Pour visualiser le déroulement des événements `KeyDown`, `KeyUp` et `KeyPress`, placez un `TextBox` sur une feuille, définissez sa propriété `Name` à `txtTexte`, puis placez le code suivant dans la section Déclarations de la fenêtre Code de la feuille :

```
 1: Private Sub txtTexte_Keydown(ByVal CodeTouche As MSForms.ReturnInteger, _
    ByVal EtatMaj As Integer)
 2:   Dim EtatTouches As String
 3:   Select Case EtatMaj
 4:     Case 0
 5:       EtatTouches = "Aucune"
 6:     Case 1
 7:       EtatTouches = "Maj"
 8:     Case 2
 9:       EtatTouches = "Ctrl"
10:     Case 3
11:       EtatTouches = "Maj et Ctrl"
12:     Case 4
13:       EtatTouches = "Alt"
14:     Case 5
15:       EtatTouches = "Maj et Alt"
16:     Case 6
17:       EtatTouches = "Ctrl et Alt"
18:     Case 7
19:       EtatTouches = "Maj, Ctrl et Alt"
20:   End Select
21:   Debug.Print ("Evénement KeyDown détecté. Code touche = " & _
       CodeTouche & ". Touches enfoncées : " & EtatTouches)
22: End Sub


23: Private Sub txtTexte_KeyUp(ByVal CodeTouche As MSForms.ReturnInteger, _
     ByVal EtatMaj As Integer)
24:   Debug.Print ("Evénement KeyUp détecté")
25: End Sub
```

```
26: Private Sub TxtTexte_Keypress(ByVal codeANSI As MSForms.ReturnInteger)
27:   Debug.Print ("Evénement KeyPress détecté. Code Ansi = " & _
      codeANSI & " = " & Chr(codeANSI))
28: End Sub
```

Testez différentes combinaisons clavier dans la zone de texte du contrôle, puis cliquez sur la case de fermeture de la feuille. Affichez la fenêtre Exécution (Ctrl+G). Le détail des événements détectés y est inscrit figure 14-8).

La première procédure est déclenchée lorsque l'événement `KeyDown` est détecté sur `txtTexte`. Une instruction `Select Case…End Select` teste la valeur de `EtatMaj` (lignes 3 à 20) et affecte une chaîne de caractères à `EtatTouches` en fonction du résultat. L'instruction de la ligne 21 inscrit dans la fenêtre Exécution de la feuille les informations ainsi récoltées.

La deuxième procédure est déclenchée lorsque l'événement `KeyUp` est détecté sur `txtTexte`. Une chaîne est alors insérée dans la fenêtre Exécution de la feuille (ligne 24).

La troisième procédure est déclenchée lorsque l'événement `KeyPress` est détecté sur `txtTexte`. L'instruction de la ligne 27 insère dans la fenêtre Exécution une chaîne suivie de la valeur de `codeANSI` et du caractère correspondant.

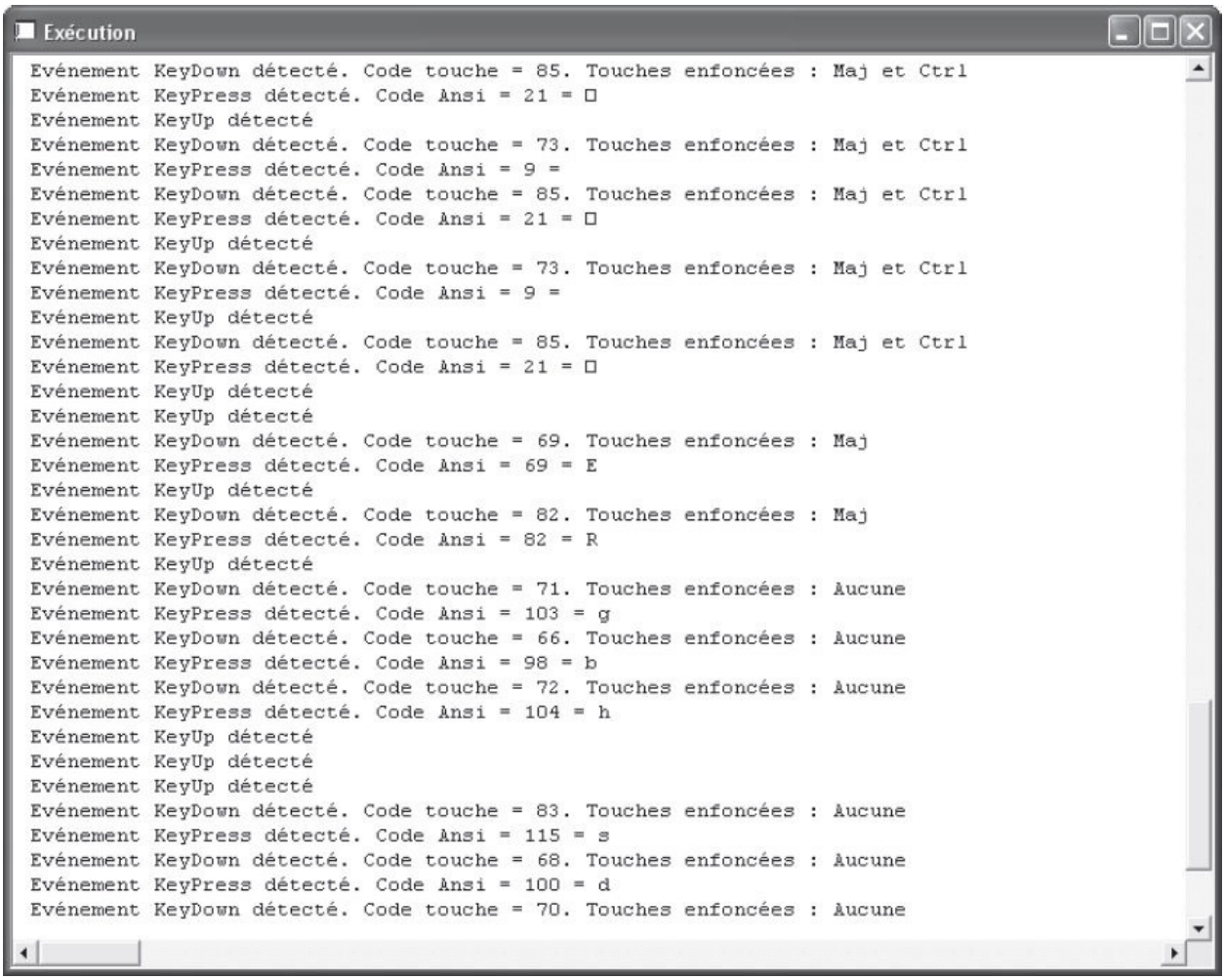


Figure 14-8 – *Les événements clavier sont précisément détectés par un programme VBA.*

- **`MouseDown`** est détecté lorsque l'utilisateur enfonce un bouton de souris. Il répond à la syntaxe suivante :

```
Private Sub Contrôle_MouseDown(ByVal Bouton As fmButton,
ByVal EtatMaj As fmShiftState, ByVal X As Single, ByVal Y As Single)
```

  `Bouton` indique quel bouton a été enfoncé : 1 pour le gauche, 2 pour le droit, 4 pour celui du centre, ces valeurs s'additionnant quand plusieurs boutons sont utilisés simultanément. `EtatMaj` reflète l'état des touches Maj, Ctrl et Alt ; les valeurs sont les mêmes que pour `KeyDown` et `KeyUp`. `X` et `Y` indiquent respectivement les distances horizontale et verticale du curseur par rapport à l'angle supérieur gauche de la feuille, au moment du clic de souris.

- **`MouseMove`** est détecté lorsque l'utilisateur déplace la souris. La syntaxe est la même que pour `MouseDown`.

- **MouseUp** est détecté lorsque l'utilisateur relâche un bouton de souris. La syntaxe est la même que pour MouseDown.
- **SpinDown** est détecté lorsqu'un clic est effectué sur le bouton inférieur ou gauche d'un SpinButton. La propriété Value du contrôle est alors décrémentée de la valeur de SmallChange, sans dépasser Min.
- **SpinUp** est détecté quand on clique sur le bouton supérieur ou droit d'un SpinButton. La propriété Value du contrôle est alors incrémentée de la valeur de SmallChange, sans dépasser Max.

Exemples d'exploitation des contrôles

Cette section présente les spécificités des contrôles qui sont à votre disposition dans la boîte à outils et les méthodes permettant de les exploiter pleinement.

Label

Le texte d'un Label n'est pas modifiable par l'utilisateur. Cependant, il est parfois utile de le changer afin de créer une interface utilisateur dynamique. Il suffit pour cela de placer des instructions redéfinissant la valeur Caption du contrôle lorsqu'un événement spécifique survient. Vous pouvez ainsi placer un seul contrôle sur une feuille, destiné à remplir une fonction variable selon les informations entrées par l'utilisateur ; c'est alors le Label l'identifiant qui indiquera sa fonction à l'utilisateur.

Dans l'exemple suivant, un Label est affecté à une zone de texte, destinée à recevoir le nom d'un nouveau membre. Deux boutons d'option, respectivement libellés "Masculin" et "Féminin", servent à spécifier le sexe du membre, le premier étant actif par défaut. Des procédures événementielles sont attachées à chacun des contrôles, afin que l'activation de l'un ou l'autre des boutons d'option modifie la propriété Caption du Label.

Pour réaliser cet exemple, créez une nouvelle feuille et placez-y un Frame contenant deux OptionButton, deux TextBox associés à un Label, et deux CommandButton. Définissez ainsi leurs propriétés :

Propriété	Valeur
Feuille	
Name	fmLabel
Caption	Modification d'un contrôle Label
Contrôle *Frame*	

Name	frSexe
Caption	Sexe du nouveau membre
Contrôle *OptionButton*	
Name	optMasculin
Caption	Masculin
Value	True
Contrôle *OptionButton*	
Name	optFeminin
Caption	Féminin
Value	False
Contrôle *Label*	
Name	lbPrenom
Caption	Prénom
Contrôle *TextBox*	
Name	txtPrenom
Value	
Contrôle *Label*	
Name	lbNom
Caption	Nom
Contrôle *TextBox*	
Name	txtNom
Value	
Contrôle *CommandButton*	
Name	cmdOK
Caption	OK
Contrôle *CommandButton*	
Name	cmdAnnuler
Caption	Annuler

Placez ensuite le code suivant dans la section Déclarations de la fenêtre Code de la feuille :

```
Sub optFeminin_Click()
  lbNom.Caption = "Nom de jeune fille"
End Sub


Sub optMasculin_Click()
  lbNom.Caption = "Nom"
End Sub


Sub cmdOK_Click()
  fmLabel.Hide
End Sub
```

Sélectionnez ensuite la feuille, puis cliquez sur le bouton Exécuter de la barre d'outils Standard. Lorsque vous cliquez sur l'un des boutons d'option, le libellé de `lbNom` varie en fonction de l'option sélectionnée (voir figure 14-9).

Attention

Lorsque vous écrivez du code modifiant la valeur `Caption` d'un `Label`, veillez à ce que la taille du contrôle permette l'affichage complet du nouveau libellé.

Figure 14-9 – *Faites varier le libellé d'un contrôle Label de façon à refléter l'information attendue.*

Contrôle TextBox

Lorsque vous utilisez un `TextBox` destiné à recevoir une information bien spécifique, vérifiez que l'utilisateur entre une valeur valide. Par exemple, si vous attendez une valeur représentant une somme à payer, assurez-vous que l'information entrée correspond bien à un nombre et qu'elle s'inscrit bien entre les limites éventuellement définies.

Pour vérifier la valeur d'un `TextBox` au moment où l'utilisateur la saisit, attachez du code à son événement `Change`. Il est parfois aussi nécessaire de vérifier qu'une valeur a bien été entrée au moment de la validation des informations entrées par l'utilisateur. Utilisez les fonctions Visual Basic contrôlant les types de données.

Rappel

Pour un rappel des fonctions contrôlant les types de données, reportez-vous au tableau 6-3.

Astuce

Pour vérifier qu'une valeur est une chaîne de caractères, utilisez la fonction `IsNumeric` qui doit alors renvoyer la valeur `False`.

Pour réaliser l'exemple suivant, créez une feuille présentant un `Label`, un `TextBox` et un `CommandButton` (voir figure 14-10) dotés des propriétés suivantes :

Propriété	Valeur
Feuille	
`Name`	`fmMaFeuille`
`Caption`	Vérification des données
Contrôle *`Label`*	
`Name`	`lbValeur`
`Caption`	Entrez la somme à traiter :
Contrôle *`TextBox`*	
`Name`	`txtValeur`
`Value`	
Contrôle *`CommandButton`*	
`Name`	`cmdOK`
`Caption`	OK

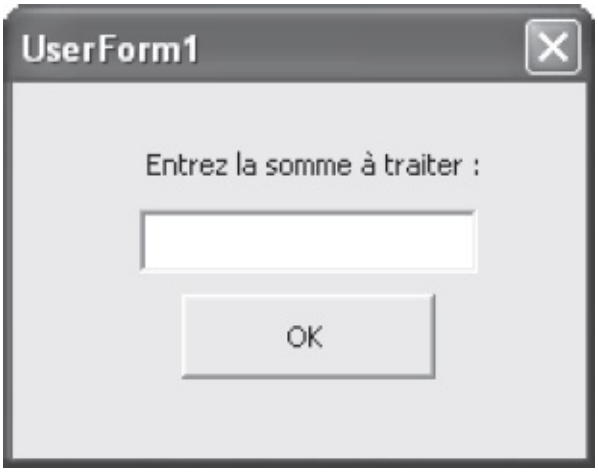


Figure 14-10 – *Votre feuille terminée.*

Placez ensuite le code suivant dans la zone Déclarations de sa fenêtre Code :

```
Private Sub txtValeur_AfterUpdate()
  If IsNumeric(txtValeur.Value)=False Then
    MsgBox txtValeur.Value & " n'est pas une valeur valide. Entrez une valeur valide", _
      vbExclamation, "Valeur non valide"
    txtValeur.Value = ""
  End If
End Sub


Private Sub cmdOK_Click()
  If txtValeur.Value="" Then
    MsgBox txtValeur.Value & "Vous devez entrer une valeur dans la zone de texte.", _
      vbExclamation, "Valeur requise"
  Else
    fmMaFeuille.Hide
  End If
End Sub
```

La première procédure utilise la fonction `IsNumeric` dans une instruction conditionnelle `If…End If` pour s'assurer que la valeur entrée dans la zone de texte est une valeur numérique ou une chaîne vide. Si tel n'est pas le cas, une boîte de dialogue s'affiche à l'attention de l'utilisateur (voir figure 14-11) et la propriété `Value` du contrôle reçoit une chaîne vide.

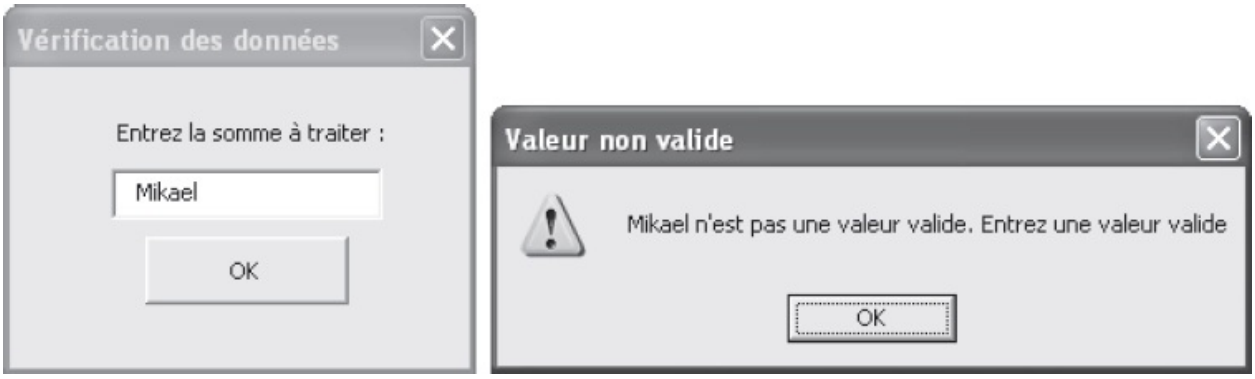


Figure 14-11 – *Une valeur non valide a été saisie dans la zone de texte.*

La procédure `cmdOK_Click` vérifie qu'une valeur a bien été entrée dans la zone de texte. Si tel n'est pas le cas, une boîte de dialogue s'affiche à l'attention de l'utilisateur (voir figure 14-12). Si, au contraire, la zone de texte contient une valeur, alors la méthode `Hide` est appliquée à la feuille afin de la masquer.

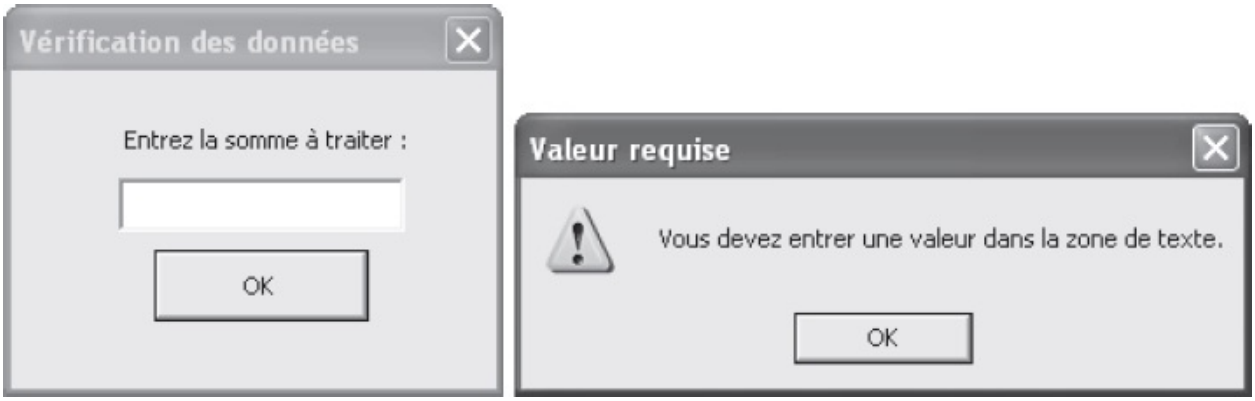


Figure 14-12 – *Une valeur doit être saisie dans la zone de texte.*

Info

Notez que la procédure `Cmd_Click` ne vérifie pas que la valeur de la zone de texte est valide (de type numérique), mais uniquement qu'elle est différente d'une chaîne vide. En effet, c'est la procédure `txtValeur_Change` qui s'en charge.

ComboBox

Les `ComboBox` sont très fréquents dans les interfaces utilisateur de la plupart des logiciels. Ils présentent en effet l'avantage d'afficher un grand nombre d'options en n'occupant qu'un espace très limité sur la feuille.

Par défaut, un `ComboBox` se comporte à la manière d'un `TextBox`, c'est-à-dire que

l'utilisateur est autorisé à saisir une valeur ne figurant pas parmi les options de la liste déroulante. Vous pouvez cependant déterminer un comportement analogue à celui d'un `ListBox`, c'est-à-dire interdire la saisie de nouvelles valeurs. Définissez pour cela la propriété `Style` du contrôle à `fmStyleDropDownList` (voir chapitre précédent).

Ajout d'éléments à la liste d'un ComboBox

La méthode *`AddItem`*

La liste affectée à un `ComboBox` est généralement déterminée dans le code du programme, à l'aide de la méthode `AddItem`, selon la syntaxe suivante :

```
Contrôle.AddItem(ElementAjoutéALaListe, Index)
```

`ElementAjoutéALaListe` est une chaîne de caractères et `Index` est la position de l'élément dans la liste (0 pour le premier élément). Ce dernier argument est généralement omis ; l'élément ajouté est alors placé en dernière position.

> **Info**
>
> Lorsque `Index` est omis, `ElementAjoutéALaListe` peut être placé directement derrière `AddItem`, en ignorant les parenthèses.

Les instructions définissant les éléments de la liste d'un `ComboBox` sont généralement placées dans une procédure événementielle attachée à l'événement `Initialize` de la feuille. La liste est ainsi mise à jour au moment de l'affichage de cette dernière.

Pour réaliser l'exemple suivant, placez un `Label`, un `ComboBox` et un `CommandButton` sur une feuille, dotés des propriétés suivantes :

Propriété	Valeur
Feuille	
`Name`	`fmComboBox`
`Caption`	Utilisation d'un contrôle `ComboBox`
Contrôle *`Label`*	
`Name`	`lbComboBox`
`Caption`	Sélection d'une valeur
Contrôle *`ComboBox`*	
`Name`	`cbComboBox`
`Value`	

Contrôle *CommandButton*	
Name	cmdQuitter
Value	Quitter

Ajoutez le code suivant :

```
Private Sub UserForm_Initialize()
  Dim mavar
  For mavar = 1 To 10
    cbComboBox.AddItem "Element de liste " & mavar
  Next mavar
End Sub


Private Sub cmdQuitter_Click()
  fmComboBox.Hide
End Sub
```

Exécutez ce code. Déroulez la liste modifiable pour visualiser les options disponibles (voir figure 14-13). Pour fermer la feuille, cliquez sur le bouton Quitter.

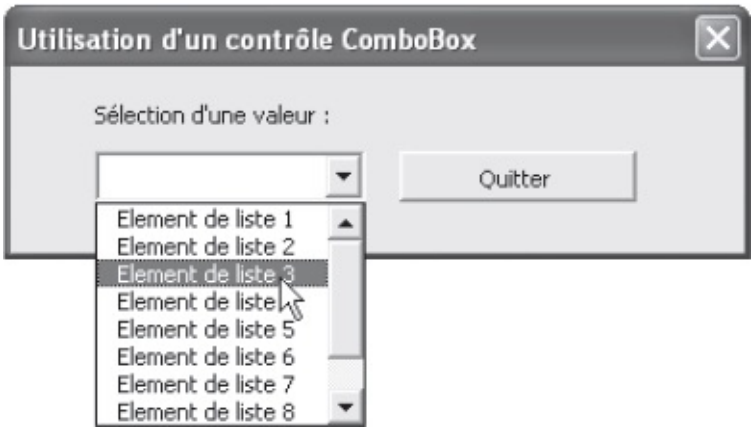


Figure 14-13 – *La liste est affectée au contrôle à l'affichage de la feuille.*

La procédure `UserForm_Initialize` est exécutée avant l'affichage de la feuille. La boucle `For…Next` s'exécute alors, ajoutant dix éléments à la liste de `cbComboBox`.

Info

Utilisez la méthode `RemoveItem` pour supprimer des éléments d'une liste.

La propriété *RowSource*

Les éléments de la liste peuvent aussi être affectés au contrôle grâce à `RowSource`. Cette propriété accepte pour valeur une chaîne représentant une cellule ou une plage de cellules. Cette possibilité est très pratique lorsqu'un projet est affecté

à un document contenant des données variables. La liste reflète ainsi toujours les données de la feuille lorsque celles-ci sont modifiées.

Dans l'exemple suivant, une feuille VBA est créée permettant à l'utilisateur d'entrer des informations sur les chiffres des différents vendeurs d'une société. Un `ComboBox` sert à sélectionner le vendeur. Pour réaliser cet exemple, créez une feuille de calcul Excel et saisissez-y les données apparaissant à la figure 14-14.

	A	B	C
1	**Vendeur**	**Région**	**Ville**
2	Mary	Nord	Lille
3	du Château	Ile de France	Paris
4	Bidault	Ouest	Quimper
5	Des Neury	Sud Ouest	Biarritz
6	Gas quai	Sud Est	Marseille

Figure 14-14 – *Les vendeurs sont répertoriés dans la colonne A.*

Accédez ensuite à Visual Basic Editor (Alt+F11) et sélectionnez le projet attaché au nouveau document dans l'Explorateur de projet. Créez une feuille VBA et placez-y un `Label`, un `ComboBox` et un `CommandButton` avec les propriétés suivantes :

Propriété	Valeur
Feuille	
`Name`	`fmRowSource`
`Caption`	Utilisation de la propriété `RowSource`
Contrôle *`Label`*	
`Name`	`lbComboBox`
`Caption`	Sélectionnez un vendeur
Contrôle *`ComboBox`*	
`Name`	`cbComboBox`
`Value`	
`RowSource`	A2:A6
Contrôle *`CommandButton`*	
`Name`	`cmdQuitter`
`Value`	Quitter

La définition de la propriété `RowSource` affecte le contenu des cellules A2 à A6 de la feuille Excel à la liste du contrôle.

Ajoutez le code suivant dans la section Déclarations de la fenêtre Code de la feuille :

```
Private Sub cmdQuitter_Click()
  fmRowSource.Hide
End Sub
```

Exécutez la feuille et déroulez la liste pour en visualiser le contenu (voir figure 14-15).

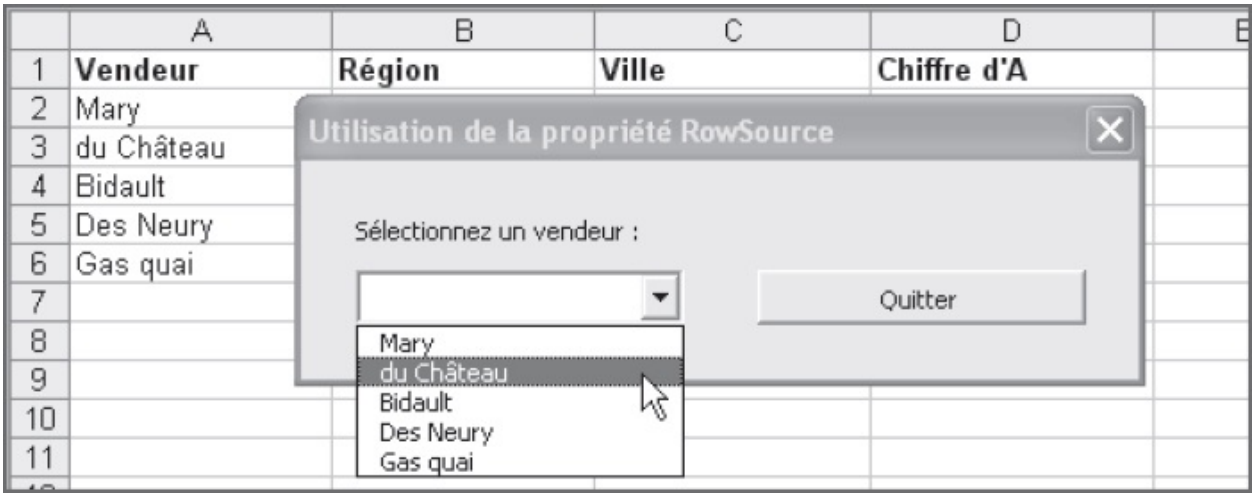


Figure 14-15 – *La liste déroulante reflète le contenu des cellules A2 à A6.*

Le `ComboBox` reflète ainsi le contenu des cellules A2 à A6. Si vous ajoutez des vendeurs (en A7, puis A8, etc.), ils n'apparaîtront pas dans la liste déroulante du contrôle. Pour remédier à ce problème, il faut définir une procédure qui affecte à `RowSource` une plage de cellules variable, reflet des cellules contenant des informations. Redéfinissez la propriété `RowSource` du `ComboBox` à une chaîne vide, puis ajoutez la procédure suivante dans la section Déclarations de la fenêtre Code de la feuille :

```
Private Sub UserForm_Initialize()
  Dim DerCell As String
  DerCell = Range("A1").End(xlDown).Address
  cbComboBox.RowSource = "A2:" & DerCell
End Sub
```

Cette procédure se déclenche à l'affichage de la feuille. L'expression `Range("A1").End(xlDown).Address` renvoie l'adresse de la dernière cellule non vide sous A1, qui est affectée à la variable `DerCell` précédemment déclarée. Enfin, la propriété `RowSource` de `cbComboBox` reçoit la plage `A2:DerCell`.

Ajoutez de nouveaux noms à la liste des vendeurs, puis exécutez à nouveau la feuille. La liste déroulante affiche la totalité des vendeurs (voir figure 14-16).

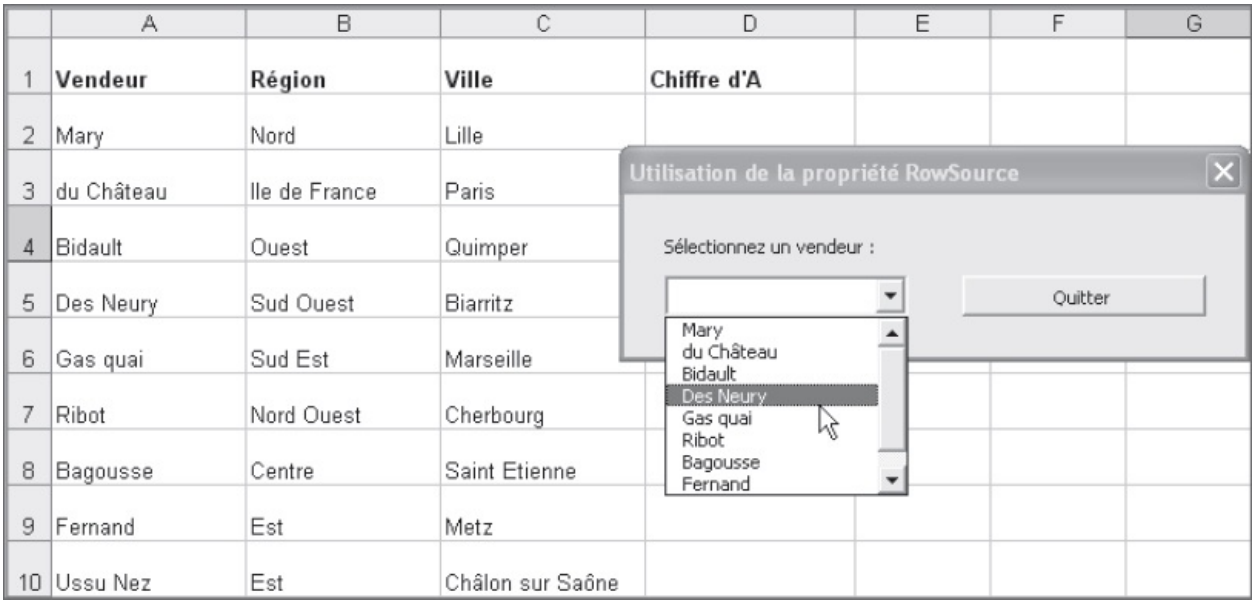


Figure 14-16 – *La liste affiche toujours la liste de tous les vendeurs.*

Valeur renvoyée par un ComboBox

La valeur sélectionnée par l'utilisateur dans un `ComboBox` est définie par sa propriété `Text` ou `Value`. `Text` renvoie le libellé apparaissant dans la zone d'édition. La valeur renvoyée par `Value` dépend de la propriété `BoundColumn` du contrôle. Si cette dernière est définie à 0, la valeur de `ListIndex` est affectée à `Value`. Autrement dit, `Value` prendra la valeur 0 si le premier élément de la liste est sélectionné, 1 si le deuxième élément est sélectionné, etc.

Dans le cas d'un `ComboBox` à plusieurs colonnes, la valeur de `BoundColumn` détermine la colonne dont le contenu sera affecté à la propriété `Value`. Par exemple, si `BoundColumn` est définie à 3, `Value` renverra une chaîne correspondant au libellé apparaissant dans la troisième colonne de la ligne sélectionnée. Pour plus d'informations sur la création de `ComboBox` à plusieurs colonnes, consultez l'Aide VBA.

Info

Utilisez la propriété `ControlSource` pour affecter la valeur sélectionnée par l'utilisateur dans un `ComboBox` à une cellule d'une feuille de classeur.

ListBox

Les contrôles `ListBox` et `ComboBox` partagent nombre de propriétés (`RowSource`, `ControlSource`, `BoundColumn`, etc.) et de méthodes (`AddItem` et `RemoveItem`, par exemple). Pour affecter une liste à un `ListBox`, utilisez l'une des méthodes présentées pour les `ComboBox`.

Un `ListBox` peut n'autoriser la sélection que d'un élément de la liste, ou autoriser des choix multiples (avec différentes méthodes de sélection).

Le type de sélection d'un `ListBox` est déterminé par sa propriété `MultiSelect`, qui accepte pour valeur l'une des constantes `fmMultiSelect` :

- **`fmMultiSelectSingle`** (par défaut). Un seul élément peut être sélectionné.
- **`fmMultiSelectExtended`.** Plusieurs éléments peuvent être sélectionnés avec les touches Maj ou Ctrl. Un clic sur un élément de la liste, sans qu'aucune de ces touches ne soit enfoncée, désélectionne tous les autres éléments.
- **`fmMultiSelectMulti`.** Plusieurs éléments peuvent être sélectionnés ou désélectionnés par de simples clics, sans modifier l'état des autres éléments de la liste.

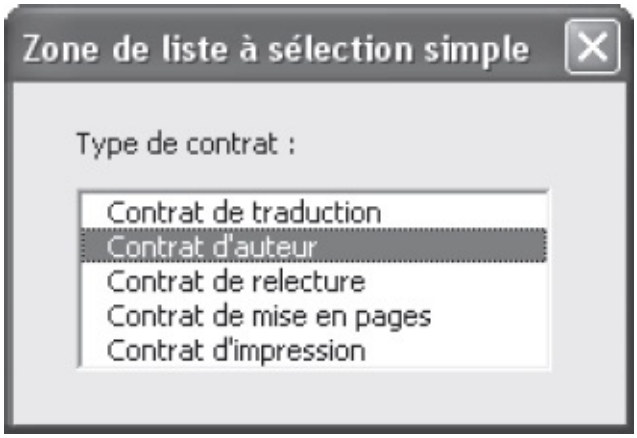


Figure 14-17 – *Une zone de liste n'autorisant qu'une sélection unique.*

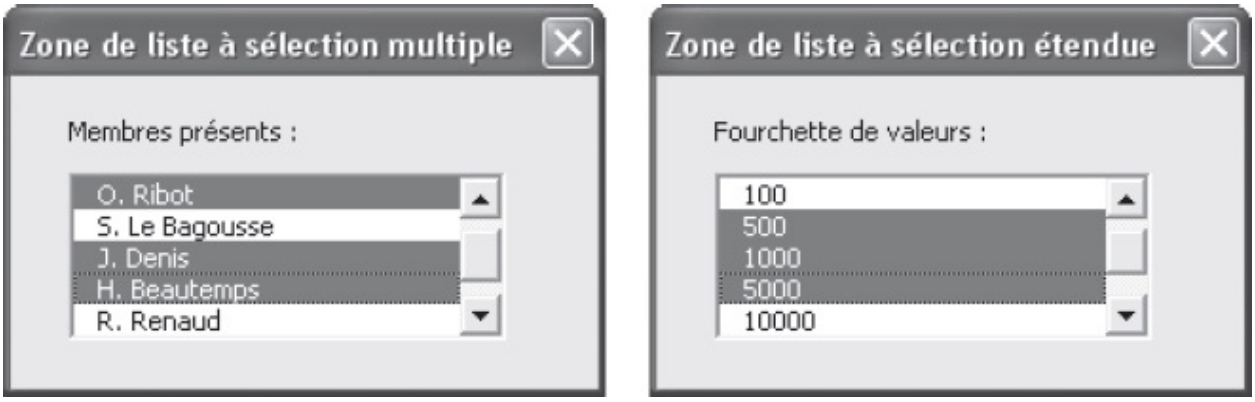


Figure 14-18 – *Une zone de liste autorisant des sélections multiples.*

Valeur renvoyée par un ListBox

La valeur d'un `ListBox` à sélection unique peut être renvoyée par sa propriété `Value` ou `Text`. Dans le cas d'un contrôle autorisant des sélections multiples, ces propriétés renvoient une chaîne vide et vous devez tester l'état de chacune des options de la liste (propriété `Selected`), selon la syntaxe suivante :

```
ContrôleListBox.Selected(IndexElement) = ValeurBooléenne
```

L'exemple suivant inscrit dans la fenêtre Exécution les éléments sélectionnés dans une liste. Placez un `Label`, un `ListBox` et un `CommandButton` sur une feuille, dotés des propriétés suivantes :

Propriété	Valeur
Feuille	
`Name`	`fmListBox`
`Caption`	Valeurs sélectionnées dans la liste
Contrôle *`Label`*	
`Name`	`lbMembres`
`Caption`	Membres présents
Contrôle *`ListBox`*	
`Name`	`cbMembres`
`Value`	
`MultiSelect`	`fmMultiSelectMulti`
Contrôle *`CommandButton`*	
`Name`	`cmdValider`
`Caption`	Valider

Ajoutez le code suivant :

```
Private Sub UserForm_Initialize()
```

```
  cbMembres.AddItem "Bidault"
  cbMembres.AddItem "Deschamps"
  cbMembres.AddItem "Kervarrec"
  cbMembres.AddItem "Goraguer"
  cbMembres.AddItem "Lemaire"
  cbMembres.AddItem "Leroux"
  cbMembres.AddItem "Martin"
  cbMembres.AddItem "Opéra"
  cbMembres.AddItem "Otello"
End Sub


Private Sub cmdValider_Click()
  Dim compteur As Single
  For compteur = 0 To (cbMembres.ListCount-1)
    If cbMembres.Selected(compteur)=True Then
      Debug.Print cbMembres.List(compteur)
    End If
  Next compteur
  fmListBox.Hide
End Sub
```

La première procédure affecte une liste au `ListBox` à l'affichage de la feuille. La procédure `cmdValider_Click` utilise les propriétés suivantes du `ListBox` :

- **`ListCount`** renvoie le nombre d'éléments de la liste.
- **`Selected(`*`index`*`)`** renvoie une valeur booléenne indiquant si l'élément à la position *`index`* est sélectionné.
- **`List(`*`index`*`)`** renvoie la chaîne de caractères correspondant à l'élément à la position *`index`*.

Une boucle `For…Next` est utilisée pour tester la valeur de chacun des éléments de la liste (le premier ayant pour index 0, l'index du dernier est égal au nombre total d'éléments de la liste, moins 1). Si l'élément testé est sélectionné, l'instruction `Debug.Print` en affiche le nom dans la fenêtre Exécution de Visual Basic Editor. Enfin, la méthode `Hide` est appliquée à la feuille afin de le fermer.

Exécutez la feuille. Sélectionnez les éléments de votre choix dans la liste, puis cliquez sur le bouton Valider. La feuille se ferme. Affichez la fenêtre Exécution de Visual Basic Editor (Ctrl+G). Les éléments sélectionnés dans la liste y sont inscrits (voir figure 14-19).

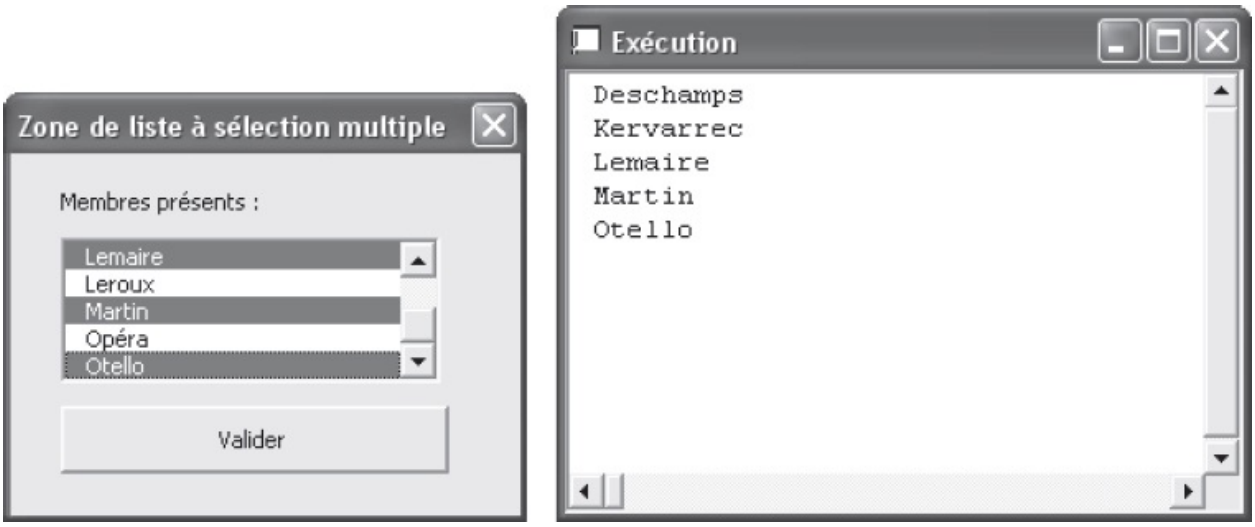


Figure 14-19 – *Testez un à un les éléments d'une liste à sélection multiple, afin de déterminer ceux sélectionnés par l'utilisateur.*

CheckBox et OptionButton

Le `CheckBox` est d'une utilisation simple. Pensez à utiliser ses propriétés `Enabled` et `Visible` pour en modifier l'accessibilité, en fonction des informations entrées par l'utilisateur. Vous pouvez aussi modifier la propriété `Caption`, afin d'en faire varier la fonction. Pour un exemple d'utilisation de ce contrôle, reportez-vous à la section consacrée à la propriété `Enabled`, au chapitre précédent.

Pour associer des `OptionButton`, soit vous les placez sur un même `Frame`, soit vous leur affectez une même propriété `GroupName`. Reportez-vous à la section dédiée à cette propriété. Pour un exemple d'exploitation des `OptionButton`, reportez-vous à la section consacrée à la propriété `Visible`.

ScrollBar

L'exemple suivant utilise un `TextBox` et un `ScrollBar` pour permettre à l'utilisateur de rechercher les années bissextiles comprises entre l'an 0 et l'an 3000. L'utilisateur peut se déplacer par cent années en cliquant dans la barre de défilement. Placez un `TextBox` et un `ScrollBar` sur une feuille, avec les propriétés suivantes :

Propriété	Valeur
Feuille	
`Name`	`fmAnnéesBissextiles`

Caption	Les années bissextiles
Contrôle *TextBox*	
Name	txtAnnée
Value	L'an 2000 est une année bissextile
Locked	True
Contrôle *ScrollBar*	
Name	scrAnnée
Value	2000
Min	0
Max	3000
SmallChange	4
LargeChange	100

Ajoutez le code suivant dans la section Déclarations de la fenêtre Code de la feuille :

```
Private Sub scrAnnée_Change()
  Dim varAnnée As Single
  varAnnée = scrAnnée.Value
  txtAnnée.Value = "L'an " & varAnnée & " est une année bissextile."
End Sub
```

La procédure `scrAnnée_Change` sera déclenchée chaque fois que l'utilisateur modifiera l'emplacement du curseur sur le `ScrollBar`. Elle affecte à `varAnnée` la valeur définie par le curseur (propriété `Value` de `scrAnnée`). La propriété `Value` du `TextBox` est ensuite définie et la zone d'édition affiche que *varAnnée* est une année bissextile.

Exécutez la feuille (figure 14-20). La barre de défilement modifie l'année affichée dans la zone de texte. Lorsque vous cliquez sur l'une des flèches de défilement, la valeur du contrôle est incrémentée ou décrémentée de 4 (`SmallChange`). Lorsque vous cliquez dans la barre, entre le curseur et une des flèches, cette valeur est incrémentée ou décrémentée de 100 (`LargeChange`).

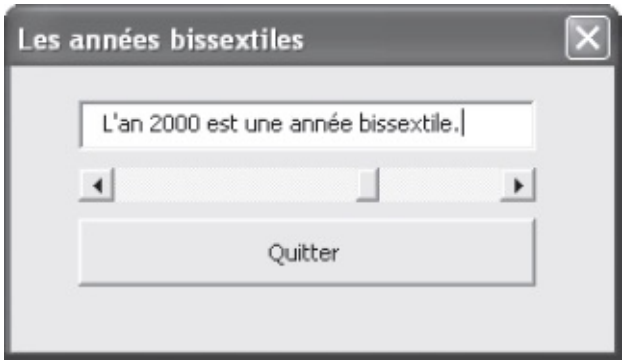


Figure 14-20 – *Pour atteindre une année éloignée de la valeur actuelle, déplacez-vous 100 ans par 100 ans, en cliquant à l'intérieur de la barre plutôt que sur les flèches de défilement.*

Ce programme fonctionne bien tant que l'utilisateur déplace le curseur en cliquant sur l'une des flèches de la barre de défilement, ou à l'intérieur de celle-ci. En revanche, s'il fait glisser le curseur, la valeur de `scrAnnée` n'est plus maîtrisée par le programme et l'utilisateur peut sélectionner une année qui n'est pas bissextile.

Partant du constat que toute année bissextile divisée par 4 est égale à un nombre entier et que tout nombre multiple de 4 correspond à une année bissextile, vous pouvez exploiter la condition suivante pour vérifier la valeur du contrôle :

```
Int(scrAnnée.Value/4) = scrAnnée.Value/4
```

Modifiez la procédure `scrAnnée_Change` de la façon suivante :

```
Private Sub scrAnnée_Change()
  While Not Int(scrAnnée.Value/4)=scrAnnée.Value/4
    scrAnnée.Value = scrAnnée.Value - 1
  Wend
  Dim varAnnée As Single
  varAnnée = scrAnnée.Value
  txtAnnée.Value = "L'an " & varAnnée & " est une année bissextile."
End Sub
```

La boucle `While…Wend` décrémente la valeur de `scrAnnée` tant que la condition posée n'est pas respectée (`While Not`) – la boucle s'exécute donc au maximum trois fois. Ainsi, si l'utilisateur opère un glisser-déplacer du curseur de défilement, nous sommes assurés que la valeur de `scrAnnée` sera modifiée si nécessaire, afin de correspondre à une année bissextile.

SpinButton

Dans l'exemple suivant, un `SpinButton` est placé à côté d'un `TextBox` et permet de sélectionner sept valeurs. Un jour de la semaine est affecté à chacune des sept valeurs possibles et affiché dans la zone de texte. Placez un `TextBox` et un `SpinButton` sur une feuille et attribuez-leur les propriétés suivantes :

Propriété	Valeur
Feuille	
`Name`	`fmJour`
`Caption`	Choix du jour

Contrôle *TextBox*	
Name	txtJour
Value	Lundi
Locked	True
Contrôle *SpinButton*	
Name	spbJour
Value	7
Min	1
Max	7
SmallChange	1

Ajoutez le code suivant dans la section Déclarations de la fenêtre Code de la feuille :

```
 1: Private Sub spbJour_Change()
 2:   Dim varBoutonToupie As Single
 3:   varBoutonToupie = spbJour.Value
 4:   txtJour.Value = QuelJour(varBoutonToupie)
 5: End Sub


 6: Private Function QuelJour(varBoutonToupie)
 7:   Select Case varBoutonToupie
 8:     Case 1
 9:       QuelJour = "Dimanche"
10:     Case 2
11:       QuelJour = "Samedi"
12:     Case 3
13:       QuelJour = "Vendredi"
14:     Case 4
15:       QuelJour = "Jeudi"
16:     Case 5
17:       QuelJour = "Mercredi"
18:     Case 6
19:       QuelJour = "Mardi"
20:     Case 7
21:       QuelJour = "Lundi"
22:   End Select
23: End Function
```

Exécutez la feuille (figure 14-21). Le bouton toupie modifie le jour affiché dans la zone de texte. Lorsqu'on arrive à Dimanche, la flèche de défilement Bas est sans effet. Lorsque Lundi est affiché, la flèche Haut est sans effet.

Figure 14-21 – *Un clic sur une flèche de défilement du contrôle augmente ou diminue sa valeur de 1 et affiche le jour correspondant.*

La procédure `spnJour_Change` se déclenche à chaque modification de la valeur de `spnJour`, c'est-à-dire à chaque fois que l'utilisateur clique sur l'une des flèches du bouton toupie. La variable `varBoutonToupie` reçoit la valeur du `SpinButton`. À la ligne 4, la propriété `Value` du `TextBox` reçoit la chaîne retournée par la fonction `QuelJour` pour l'argument `varBoutonToupie`, c'est-à-dire le texte à afficher selon la valeur de l'argument (instruction `Select Case…End Select`, lignes 7 à 22).

Nous allons maintenant modifier ce programme afin que les flèches de défilement Haut et Bas du bouton toupie permettent de passer de Lundi à Dimanche et inversement. Une technique courante pour créer ce genre de boucle consiste à autoriser une valeur minimale et une valeur maximale pour le `SpinButton`, qui ne seront pas visibles pour l'utilisateur, mais exploitées par une instruction conditionnelle. Définissez les propriétés `Min` à 0 et `Max` à 8, puis modifiez `spbJour_Change` de la façon suivante :

```
 1: Private Sub spbJour_Change()
 2:   If spbJour.Value=0 Then
 3:     spbJour.Value = 7
 4:   ElseIf spbJour.Value=8 Then
 5:     spbJour.Value = 1
 6:   End If
 7:   Dim varBoutonToupie As Single
 8:   varBoutonToupie = spbJour.Value
 9:   txtJour.Value = QuelJour(varBoutonToupie)
10: End Sub
```

Attention

Lorsque la propriété `Value` d'un `ScrollBar` ou d'un `SpinButton` est égale à sa propriété `Max` ou `Min`, redéfinir la valeur de l'une de ces dernières dans la fenêtre Propriétés entraîne aussi la redéfinition de `Value`.

Dans le cas présent, l'affectation de la valeur 8 à `Max` redéfinira aussi `Value` à 8. Vous devez par conséquent changer `Value` à 7 après avoir modifié `Max`.

La structure conditionnelle ajoutée teste la valeur de `spbJour`. Si l'utilisateur clique sur la flèche de défilement Haut du bouton toupie alors que sa valeur est 1 (la zone de texte affiche Dimanche), `Value` passe à 0 (`Min`) et la structure conditionnelle (lignes 2 et 3) la redéfinit à 7 (`Max`-1). De même, si l'utilisateur clique sur la flèche de défilement Bas alors que la valeur est 7 (la zone de texte affiche Lundi), `Value` passe à 8 (`Max`) et la structure conditionnelle (lignes 4 et 5) la redéfinit à 1 (`Min`+1).

Info

La procédure conditionnelle gérant les valeurs en boucle du bouton toupie peut tout aussi bien s'écrire ainsi :

```
If spbJour.Value=spbJour.Min Then
  spbJour.Value = spbJour.Max-1
ElseIf spbJour.Value = spbJour.Max Then
  spbJour.Value = spbJour.Min + 1
End If
```

Exploiter les informations d'une feuille VBA

Pour exploiter les informations d'une feuille VBA, vous devez les passer à une procédure d'un module de code, en spécifiant les valeurs des contrôles comme arguments.

Rappel

Les appels de procédure et le passage d'arguments ont été traités au chapitres 5.

Lorsque vous souhaitez développer une interface utilisateur pour simplifier une tâche, procédez selon les étapes suivantes :

1. Développez votre feuille. Déterminez les propriétés des différents contrôles et écrivez les procédures événementielles qui leur sont spécifiques.
2. Dans un module de code, écrivez une procédure qui affichera la feuille (`NomFeuille.Show`). Vous pourrez éventuellement l'affecter par la suite à un bouton de barre d'outils ou à une commande de menu (voir chapitres 11), de façon à simplifier l'exécution du programme.
3. Votre feuille contiendra probablement un bouton de validation (OK). La procédure événementielle pour l'événement `Click` de ce bouton devra :
   - vérifier que les informations entrées par l'utilisateur sont valides ;
   - masquer la feuille (`Me.Hide`) ;

– appeler la procédure qui traitera les informations.

La procédure événementielle suivante appelle `ValidationConditions` en lui passant pour arguments les valeurs des contrôles `TxtVolume`, `TxtPrixTrad`, `TxtPrixAudio`, `ChkDefinitif` et `ChkImprimer`.

```
Private Sub CmdOK_Click()
  'Instructions vérifiant la validité des données
  Me.Hide
  Call ValidationConditions(TxtVolume.Value, TxtPrixTrad.Value, _
     TxtPrixAudio.Value, ChkDefinitif.Value, ChkImprimer.Value)
End Sub
```

La procédure `ValidationConditions` devra se trouver dans un module de code standard. Son instruction de déclaration comprendra les arguments correspondant aux valeurs passées. Elle pourra par exemple se présenter comme suit :

```
Sub ValidationConditions(Volume, PrixTrad, Prix, DefinitifOuNon, ImprimerOuNon)
  'Instructions
End Sub
```

Conseil

Si vous devez recueillir de nombreuses informations, créez plusieurs feuilles VBA et structurez-les de la façon suivante :

1. Affichage de la première feuille.
2. Stockage de ses données dans des variables d'un module de code.
3. Masquage de la feuille
4. Affichage de la seconde feuille et retour au point 2.

Etc.

N. Appel de la procédure en charge de traiter les données entrées dans les différentes feuilles.

QUATRIÈME PARTIE

Notions avancées de la programmation Excel

15

Programmer des événements Excel

Vous avez découvert, au cours de cet ouvrage, l'environnement de Visual Basic Editor, les techniques de programmation en VBA et les outils d'aide au développement de projet. Ce chapitre vous apprendra à gérer précisément les événements utilisateur susceptibles d'affecter les objets Excel au cours d'une utilisation classique du logiciel. Vous verrez qu'il est possible de détecter les actions de l'utilisateur sur un classeur et de créer des procédures affectées à ces événements.

Au même titre que les contrôles placés sur une feuille UserForm, les classeurs et les feuilles Excel sont des objets auxquels vous pouvez affecter des procédures événementielles. Vous gérez ainsi des événements tels que la création d'un nouveau classeur, la modification d'une cellule, l'activation d'une feuille, etc.

L'objet Application

Au sommet du modèle d'objets d'Excel se trouve `Application`. Il représente l'ensemble de l'application et est donc l'objet conteneur de tous les autres. L'objet `Application` est particulièrement intéressant pour le développeur. Il intègre en effet la gestion d'événements de niveau application, susceptibles d'intervenir lors d'une utilisation courante du tableur : création, ouverture ou fermeture d'un classeur, etc.

Les sections suivantes vous indiquent comment créer des procédures événementielles pour l'objet `Application`.

Déclaration et instanciation de l'objet Application

Les procédures destinées à gérer les événements utilisateur de niveau application ne peuvent être écrites que dans un module de classe, dans lequel vous devez déclarer une variable objet de type `Application` à l'aide du mot-clé

`WithEvents`. Ce dernier indique que l'objet ainsi créé gérera les événements.

1. Affichez l'Explorateur de projet. Cliquez-droit sur l'un des éléments de Personal.xlsb et, dans le menu contextuel qui s'affiche, choisissez Insertion > Module de classe.
2. Par défaut, le nouveau module est nommé Class1 (Class2 si Class1 existe déjà, etc.). Sélectionnez-le puis appuyez sur la touche F4. Dans la fenêtre Propriétés, affectez-lui un nom représentatif (ici, `ModuleGestionEvt`).
3. Ouvrez la fenêtre Code du module et saisissez-y l'instruction suivante :

```
Public WithEvents MaVarApplication As Application
```

Info

Le mot-clé `WithEvents` n'est valide que dans un module de classe.

Notre projet intègre maintenant un module de classe dans lequel nous avons défini un objet `MaVarApplication` de type `Application`, capable de gérer les événements. Nous devons maintenant en créer une instance et lui affecter l'objet `Application` d'Excel. Cette procédure peut se trouver dans n'importe quel module de code.

Supposons ici que vous souhaitiez que la gestion des événements de niveau application soit active dès l'ouverture d'Excel. Il vous suffit d'écrire l'instruction d'instanciation de la variable dans une procédure événementielle affectée à l'ouverture du classeur PERSONAL.XLSB.

Procédez comme suit :

1. Dans l'Explorateur de projet, double-cliquez sur l'objet `ThisWorkbook` du projet PERSONAL.XLSB. Sa fenêtre Code s'affiche. Dans la section Déclaration, placez l'instruction suivante :

```
Dim MonInstance As New ModuleGestionEvt
```

   où `ModuleGestionEvt` est le nom affecté au module de classe créé dans la section précédente.
2. Dans la zone Objet de la fenêtre Code, sélectionnez `Workbook`. Les instructions d'encadrement d'une procédure événementielle de type `Open` sont automatiquement créées. Placez entre celles-ci l'instruction d'affectation de la variable `MonInstance`. La procédure doit se présenter comme suit :

```
Private Sub Workbook_Open()
  Set MonInstance.MaVarApplication = Application
End Sub
```

L'instruction affecte l'objet `Application` à la propriété `MaVarApplication` de l'objet `MonInstance`.

Vous pouvez maintenant créer des procédures événementielles de niveau application.

Création de procédures événementielles de niveau application

Les procédures événementielles de niveau application se trouveront dans le module de classe créé dans la section précédente. Leur création s'effectue de la même façon que pour les contrôles.

1. Ouvrez la fenêtre Code du module de classe `ModuleGestionEvt`.
2. Dans la zone Objet, choisissez la variable `MaVarApplication`. Les instructions d'encadrement d'une procédure événementielle de type `NewWorkbook` (événement nouveau classeur) sont automatiquement insérées. Placez une instruction d'affichage de boîte de dialogue à l'aide de la fonction `MsgBox` :

```
Private Sub MaVarApplication_NewWorkbook(ByVal Wb As Excel.Workbook)
   MsgBox "L'utilisateur a créé un nouveau classeur", vbOKOnly + vbInformation
End Sub
```

L'argument `Wb` correspond à l'objet `Workbook` créé. Vous pouvez utiliser cette variable dans votre procédure pour tester et manipuler le classeur créé. L'instruction suivante affiche une boîte de dialogue dans laquelle est mentionné le nombre de feuilles du classeur :

```
MsgBox "Le classeur contient " & Wb.Sheets.Count & " feuilles."
```

3. Enregistrez le projet et quittez Excel.
4. Relancez Excel et créez un nouveau classeur. La boîte de dialogue de la figure 15-1 s'affiche.

Figure 15-1 – *Une procédure peut être affectée à la création d'un nouveau*

classeur.

La reconnaissance de l'événement `NewWorkbook` est utile pour créer des procédures destinées à aider l'utilisateur dans ses tâches courantes. Vous pouvez par exemple afficher une feuille UserForm contenant une liste d'options de classeurs : l'utilisateur sera alors invité à indiquer le type de classeur qu'il souhaite réaliser et la procédure appelée insérera les données essentielles, créera le nombre de feuilles voulues, enregistrera le classeur dans le bon dossier, etc.

La liste Procédure recense les événements gérés par l'objet `Application`. Vous y trouverez notamment `WindowActivate` (activation d'une fenêtre par l'utilisateur), ou encore `WorkbookBeforePrint` (survient avant l'exécution d'une impression). Les événements dont le nom contient la chaîne `Before` surviennent avant l'exécution d'une tâche. Ils intègrent le plus souvent un argument `Cancel` permettant d'annuler cette dernière si les conditions requises ne sont pas remplies. Les événements dont le nom contient la chaîne `After` surviennent après l'exécution d'une tâche.

Pour des informations supplémentaires sur un événement de l'objet `Application`, choisissez-le dans la liste Procédure, sélectionnez son nom dans la fenêtre Code et appuyez sur la touche F1.

Propriétés de l'objet Application

Les propriétés de l'objet `Application` sont lisibles et modifiables comme celles de n'importe quel objet. Le tableau 15-1 présente sommairement les principales. Pour plus d'informations, saisissez le nom de la propriété dans une fenêtre Code, sélectionnez-le et tapez sur la touche F1.

Tableau 15-1. Les propriétés essentielles de l'objet Application

Propriété	Description
`ActiveWorkbook`	Renvoie l'objet `Workbook` correspondant au classeur actif (en lecture seule).
`ActiveSheet`	Renvoie l'objet `Worksheet` correspondant à la feuille active du classeur spécifié (en lecture seule).
`ActiveCell`	Renvoie l'objet `Range` correspondant à la cellule active de la feuille de classeur spécifiée (en lecture seule).
`Caption`	Renvoie le nom de l'application, qui apparaît dans la barre de titre. La valeur par défaut est "Microsoft Excel". La figure 15-2 représente une session Excel dans laquelle nous avons redéfini la valeur de la propriété `Caption` de l'objet `Application` (`Application.Caption = ChaîneTitre`).

`Cursor`	Définit l'apparence du curseur au cours des différentes phases d'une macro. Quatre constantes correspondent aux quatre curseurs disponibles. Si vous modifiez l'apparence du curseur au cours d'une macro, n'oubliez pas de redéfinir la propriété `Cursor` à `xlDefault` à la fin de l'exécution.
`DisplayAlerts`	Paramétrez cette propriété sur `False` pour éviter l'affichage de messages Excel. N'omettez pas de redéfinir la propriété `DisplayAlerts` sur `True` en fin de macro.
`DisplayFormulaBar`, `DisplayScrollBars` et `DisplayStatusBar`	Déterminent respectivement si les barres de formule, de défilement et d'état sont affichées (`True`) ou non (`False`).
`EnableCancelKey`	Détermine si l'utilisateur peut ou non interrompre une macro en cours d'exécution à l'aide de la combinaison Ctrl+Pause. Par défaut, sa valeur est `xlInterrupt`. Pour interdire l'interruption, définissez cette propriété sur `xlDisabled` ou sur `xlErrorHandler`.
`PathSeparator`	Renvoie le caractère utilisé comme séparateur dans les chemins – une barre oblique inverse (\) sous Windows et deux-points (:) sous Macintosh. Cette propriété est utile si vous développez des projets VBA tournant sur les deux systèmes.
`ScreenUpdating`	Détermine si l'affichage écran est mis à jour (`True`) ou non (`False`) lors de l'exécution d'une macro. Ne pas le mettre à jour améliore considérablement les performances. De plus, les tâches effectuées sont invisibles pour l'utilisateur tant que la macro n'est pas terminée. La propriété `ScreenUpdating` est automatiquement redéfinie à `True` lorsque la macro s'achève.
`ThisWorkbook`	Renvoie le classeur contenant le code de la macro qui s'exécute. Il ne s'agit pas forcément du classeur actif (`ActiveWorkbook`).
`UserName`	Détermine le nom d'utilisateur. Il s'agit par défaut du nom apparaissant dans l'onglet Général de la boîte de dialogue Options.

Attention

Lorsque vous modifiez les propriétés de l'objet `Application` au cours d'une macro, pensez à les redéfinir à la fin de l'exécution.

Si vous définissez `EnableCancelKey` à `False`, vous devez être certain que votre macro n'est pas boguée et gère les exceptions. Si, par exemple, elle exécute une boucle à l'infini, vous n'aurez aucun moyen de l'interrompre.

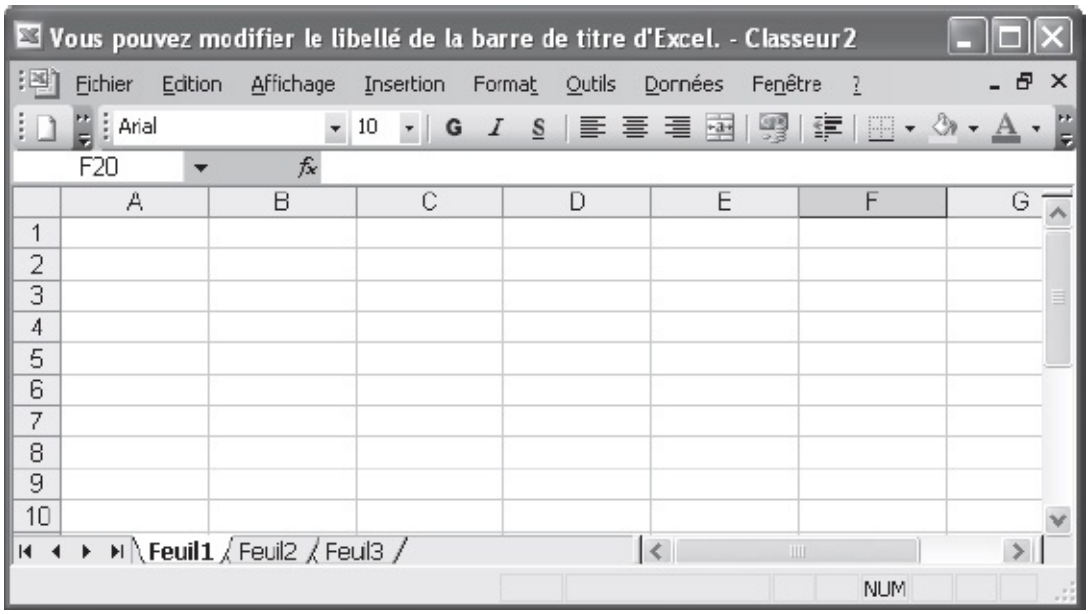

Vous pouvez modifier le libellé de la barre de titre d'Excel. - Classeur2
Fichier Edition Affichage Insertion Format Outils Données Fenêtre ?
Arial
10
F20
A B C D E F G
1 2 3 4 5 6 7 8 9 10
Feuil1 Feuil2 Feuil3
NUM

Figure 15-2 – *Vous pouvez modifier le titre de la session Excel.*

Méthodes de l'objet Application

Le tableau 15-2 présente quelques-unes des méthodes les plus intéressantes de l'objet `Application`.

Tableau 15-2. Les méthodes essentielles de l'objet Application

Méthode	Description
`Calculate`	Effectue un calcul forcé sur tous les classeurs ouverts. Utilisez cette méthode pour mettre à jour les classeurs ouverts lorsque le tableur est paramétré pour effectuer des calculs manuels (onglet Calcul de la boîte de dialogue Options).
`OnKey`	Exécute une procédure lorsqu'une combinaison de touches est activée. Pour plus d'informations, consultez l'aide en ligne de VBA.
`OnTime`	Programme l'exécution d'une procédure à un moment précis (ce soir à 20 h 30 par exemple). Pour plus d'informations, consultez l'aide en ligne de VBA.
`OnUndo`	Définit le texte placé derrière la commande Annuler du menu Édition et la procédure exécutée lorsque l'utilisateur sélectionne cette commande. Pour plus d'informations, consultez l'aide en ligne de VBA.

L'objet ThisWorkbook

Disponible dans le dossier Microsoft Excel Objets de l'Explorateur de projet, l'objet `ThisWorkbook` représente le classeur correspondant au projet affiché dans l'Explorateur de projet (voir figure 15-3). Au même titre que l'objet `Application`, il intègre des propriétés et des méthodes pour le manipuler et se renseigner sur

son état. Il prend aussi en charge un certain nombre d'événements.

Double-cliquez sur `ThisWorkbook` dans l'Explorateur de projet. Dans la zone Objet de la fenêtre Code, sélectionnez `Workbook`. Les instructions d'encadrement d'une procédure événementielle `Open` (ouverture du classeur) sont automatiquement insérées.

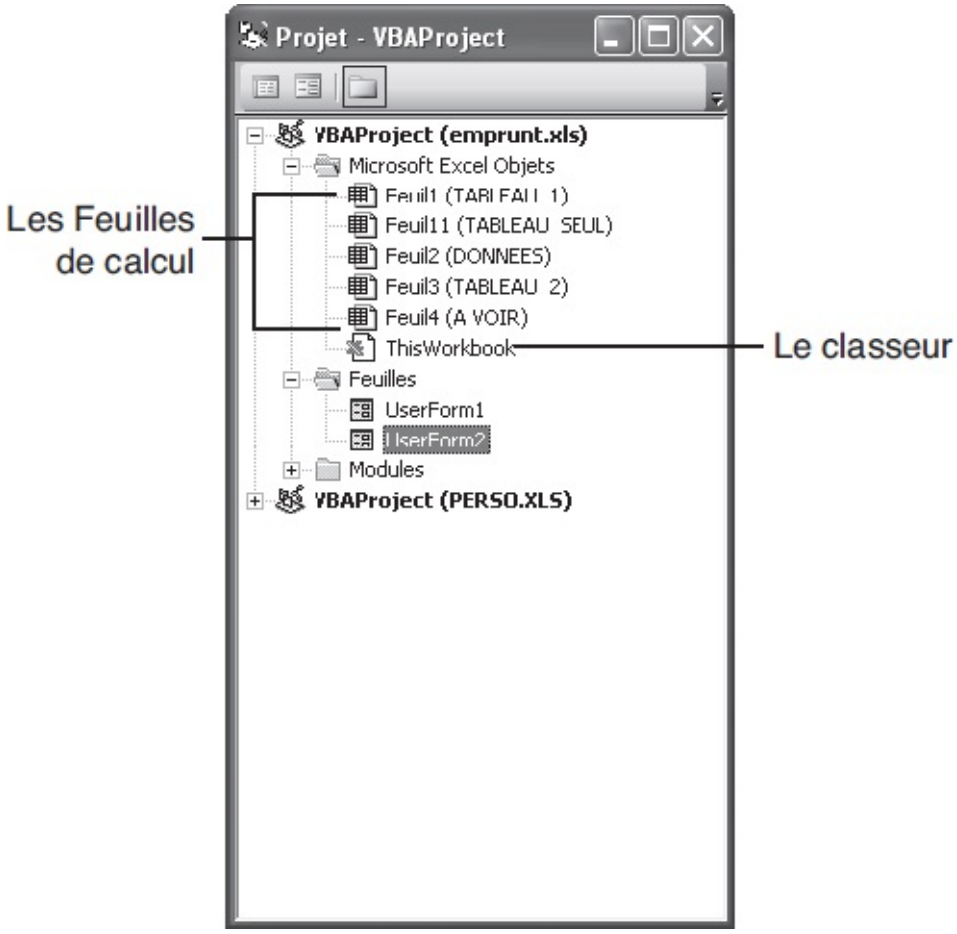


Figure 15-3 – *Les objets du dossier Microsoft Excel Objets correspondent au classeur et à ses feuilles de calcul.*

`ThisWorkbook` est un objet `Workbook` (classeur). Il possède donc les mêmes propriétés et les mêmes méthodes que tous les objets de ce type. Il prend en charge un nombre important d'événements, tels que `Open` (ouverture du classeur) ou `SheetActivate` (activation d'une autre feuille de calcul). Certains prennent des arguments précisant l'action de l'utilisateur. Par exemple, `SheetActivate` prend pour argument un objet `Worksheet` qui correspond à la feuille activée.

Pour une liste détaillée des propriétés, méthodes et événements de ces objets,

saisissez Workbook dans une fenêtre Code, sélectionnez ce mot et appuyez sur la touche F1.

Ouvrez n'importe quel classeur Excel. Ouvrez la fenêtre Code de l'objet `ThisWorkbook` du projet et placez-y le code suivant :

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
  MsgBox "La feuille " & Sh.Name & " a été activée.", _
    vbOKOnly + vbInformation, "Evénement SheetActivate détecté"
End Sub
```

Retournez dans Excel ; à chaque fois que vous activez une feuille, une boîte de dialogue affiche son nom. Cette fonction est intéressante si vous souhaitez limiter l'accès à une feuille de calcul. La procédure suivante affiche une boîte de dialogue dans laquelle l'utilisateur est invité à entrer un mot de passe s'il tente d'accéder à la feuille « Clients ».

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
  Application.EnableCancelKey = xlDisabled
  If Sh.Name="Clients" Then
    ActiveWindow.Visible = False
    Dim MotDePasse As String
    MotDePasse = InputBox("Entrez votre mot de passe.", _
      "Mot de passe requis")
    If Not MotDePasse="jk85m" Then
      MsgBox "Le mot de passe saisi est incorrect.", _
        vbOKOnly + vbInformation, "Mot de passe incorrect"
      ThisWorkbook.Sheets("Feuil2").Activate
    End If
    Windows("Representants par clients.xlsx").Visible = True
  End If
End Sub
```

Notez que nous avons défini la propriété `EnableCancelKey` de l'objet `Application` à `xlDisabled`, de façon que l'utilisateur ne puisse pas interrompre la macro. La propriété `Visible` de l'objet `ActiveWindow` (la fenêtre active) est définie à `False` de manière à masquer la feuille de classeur tant que l'utilisateur n'a pas entré le mot de passe. Si l'utilisateur saisit un mauvais mot de passe, la feuille Feuil2 du classeur est activée. Enfin, la propriété `Visible` de la fenêtre correspondant au classeur est définie à `True` de façon à afficher de nouveau le classeur.

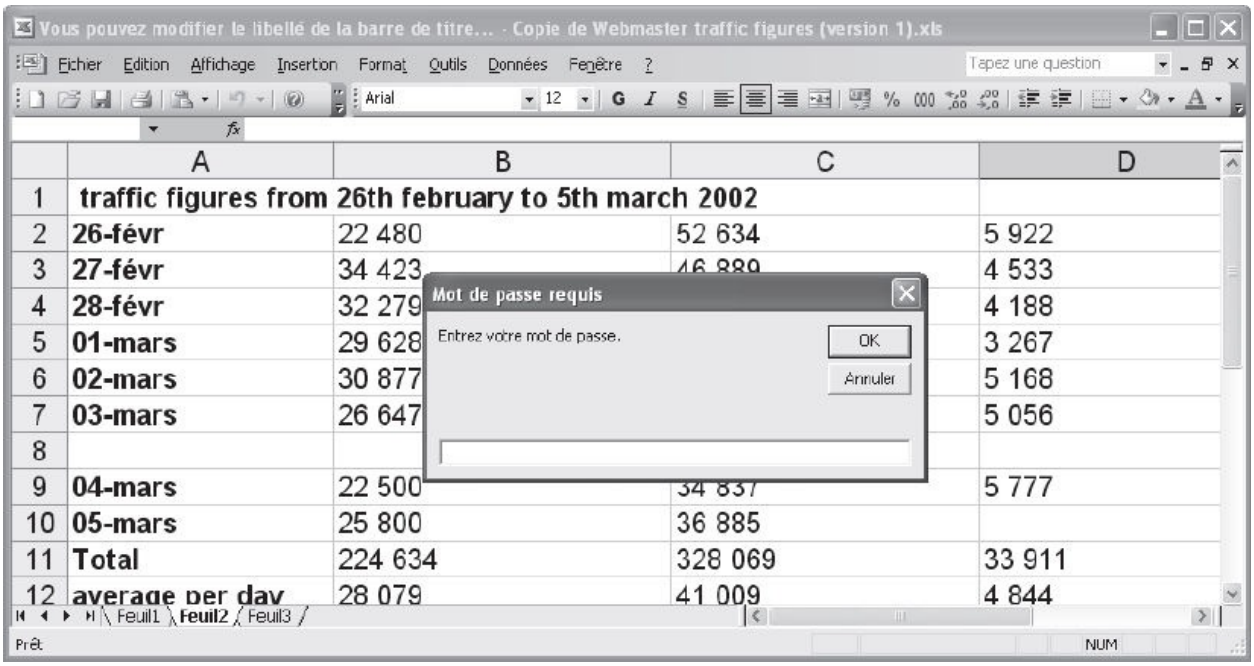

Vous pouvez modifier le libellé de la barre de titre... - Copie de Webmaster traffic figures (version 1).xls
Fichier Edition Affichage Insertion Format Outils Données Fenêtre ?
Tapez une question
Arial 12
traffic figures from 26th february to 5th march 2002
26-févr 22 480 52 634 5 922
27-févr 34 423 4 533
28-févr 32 279 4 188
01-mars 29 628 3 267
02-mars 30 877 5 168
03-mars 26 647 5 056
04-mars 22 500 5 777
05-mars 25 800 36 885
Total 224 634 328 069 33 911
average per day 28 079 41 009 4 844
Mot de passe requis
Entrez votre mot de passe.
OK
Annuler
Feuil1 Feuil2 Feuil3
Prêt
NUM

Figure 15-4 – *Vous pouvez protéger l'accès aux données d'une feuille de calcul.*

> **Rappel**
>
> Pour empêcher l'utilisateur de lire le mot de passe dans Visual Basic Editor, protégez aussi votre projet par mot de passe.

Pour des informations supplémentaires sur un événement de l'objet `Application`, choisissez-le dans la liste Procédure, sélectionnez son nom dans la fenêtre Code et appuyez sur la touche F1.

L'objet Worksheet

Le dossier Microsoft Excel Objets contient aussi des objets `Worksheet` correspondant aux feuilles du classeur (voir figure 15-3).

Les feuilles gèrent les événements présentés dans le tableau 15-3. Pour accéder aux rubriques d'aide d'un événement, ouvrez la fenêtre Code d'un objet Feuille, choisissez l'événement voulu dans la liste Procédure, sélectionnez son nom et appuyez sur la touche F1.

Tableau 15-3. Événements gérés par les objets Worksheet

Événement	Description
`Activate`	Survient lorsque la feuille de calcul est activée.

`BeforeDoubleClick`	Survient avant le double-clic sur une feuille de calcul ou un graphique. La procédure reçoit l'objet `Range` correspondant à la cellule sur laquelle l'utilisateur a double-cliqué. Utilisez l'argument `Cancel` pour annuler si nécessaire.
`BeforeRightClick`	Survient avant que le clic-droit ne soit validé. La procédure reçoit l'objet `Range` correspondant à la cellule sur laquelle l'utilisateur a cliqué-droit. Utilisez l'argument `Cancel` pour annuler si nécessaire.
`Calculate`	Survient lorsqu'un calcul est effectué. C'est le cas si Excel est paramétré pour un calcul automatique et si la valeur d'une cellule utilisée comme argument dans des fonctions de la feuille est modifiée. Si Excel est paramétré pour un calcul manuel, l'événement survient lorsque l'utilisateur force le calcul.
`Change`	Survient lorsque la valeur d'une cellule est modifiée (on considère que c'est le cas après que l'utilisateur a sélectionné une autre cellule). La procédure reçoit l'objet `Range` correspondant à la cellule dont la valeur a été changée.
`Deactivate`	Survient lorsque la feuille de calcul est désactivée, c'est-à-dire lorsque l'utilisateur en sélectionne une autre.
`SelectionChange`	Survient lorsque l'utilisateur modifie la cellule ou la plage de cellules sélectionnée. La procédure reçoit l'objet `Range` correspondant.

La procédure suivante affiche une boîte de dialogue invitant l'utilisateur à mettre à jour les classeurs liés s'il modifie la valeur de la cellule B5 de la feuille en cours. S'il clique sur le bouton Oui, `ProcédureMiseAJour` est appelée et la valeur de la cellule B5 lui est passée comme argument.

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
  If Target.Address="$B$5" Then
    Dim MiseAJour As Integer
    MiseAJour = MsgBox("Mettre à jour les classeurs liés ?", _
    vbYesNo + vbQuestion, "Données communes modifiées")
    If MiseAJour=vbYes Then
      Call ProcédureMiseAJour(Target.Value)
    End If
  End If
End Sub
```

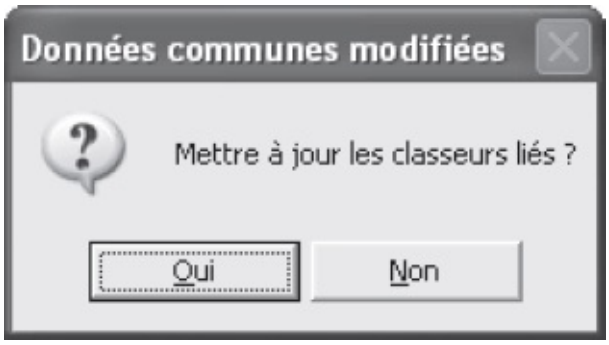


Figure 15-5 – *S'il modifie la valeur de la cellule B5, l'utilisateur est invité à mettre à jour les classeurs liés.*

Attention

Notez que les fenêtres Code des objets `Workbook` et `Worksheet` ne peuvent contenir que des procédures événementielles. Les procédures appelées doivent se trouver dans des modules de code.

Pour une liste détaillée des propriétés, méthodes et événements des feuilles, saisissez Worksheet dans une fenêtre Code, sélectionnez ce mot et appuyez sur la touche F1.

16

Protéger et authentifier des projets VBA

Les virus macros

Parmi les virus, ceux qui infectent les macros font partie des plus fréquents. Il s'agit en général de macros conçues pour s'exécuter à l'ouverture d'un document ou pour se substituer à certaines commandes de l'application hôte. Ces virus sont attachés à un document et sont susceptibles d'affecter les documents du même type sur une machine infectée.

Du fait de leur grande souplesse de personnalisation et de leur popularité, Word et Excel sont les applications les plus touchées, mais tout type de document capable de stocker des macros peut être infecté.

Les virus macros sont plus ou moins nuisibles. Les plus néfastes suppriment des fichiers du disque dur ou sur le réseau, ou endommagent des fichiers indispensables au bon fonctionnement d'une application, voire du système. Plus pernicieux, certains suppriment des données dans un document ou au contraire ajoutent des mots et en modifient ainsi le sens.

Se protéger des virus macros

Dans la version 97 d'Office, la seule option de protection consistait à prévenir l'utilisateur de la présence de macros dans un document au moment de son ouverture et à lui proposer de les désactiver ; cependant, le document était alors ouvert en lecture seule et il était impossible de lui apporter des modifications.

Depuis la version 2000 d'Office, Microsoft a introduit deux nouveautés dédiées à la sécurité et à la protection contre les virus macros : la possibilité de définir des niveaux de sécurité, comparables à ceux qui existaient déjà dans Internet Explorer, et la signature électronique des macros VBA.

Définir un niveau de sécurité

Les options de sécurité ont sensiblement été modifiées dans les versions 2007 et 2010 d'Excel. Procédez comme suit :

1. Cliquez sur l'onglet Fichier puis sur le bouton Options Excel.
2. Dans le volet gauche de la fenêtre, sélectionnez Centre de gestion de la confidentialité (voir figure 16-1).
3. Cliquez sur le bouton Paramètres de gestion du centre de confidentialité. La fenêtre qui s'ouvre est composée de plusieurs onglets. Nous présentons sommairement ceux qui nous intéressent ci-après (pour plus d'informations, consultez l'aide d'Excel) :
   – Éditeurs approuvés. Il s'agit de ceux dont vous reconnaissez les signatures numériques dignes de confiance. Les macros des documents qu'ils ont signés seront activées.

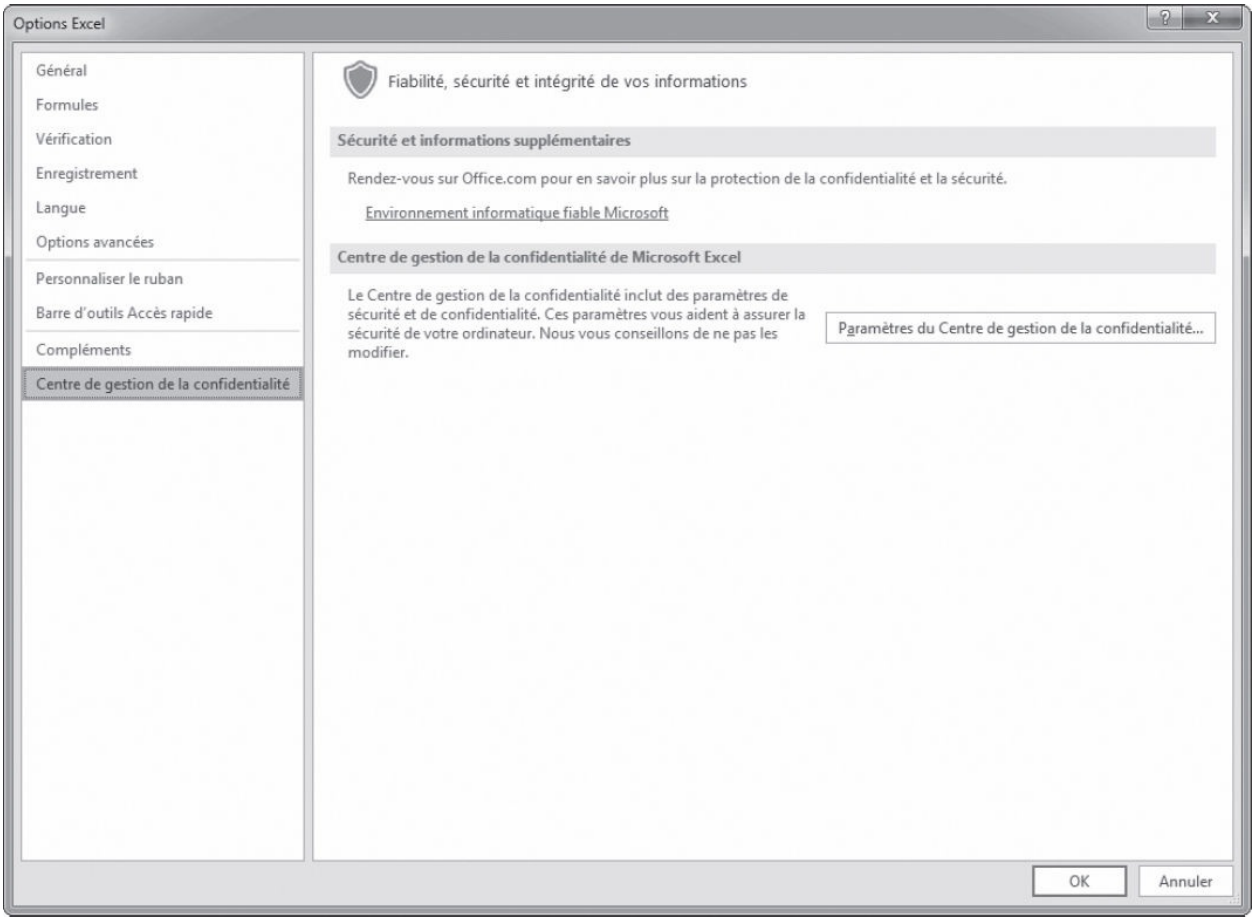

Options Excel
Général
Formules
Vérification
Enregistrement
Langue
Options avancées
Personnaliser le ruban
Barre d'outils Accès rapide
Compléments
Centre de gestion de la confidentialité
Fiabilité, sécurité et intégrité de vos informations
Sécurité et informations supplémentaires
Rendez-vous sur Office.com pour en savoir plus sur la protection de la confidentialité et la sécurité.
Environnement informatique fiable Microsoft
Centre de gestion de la confidentialité de Microsoft Excel
Le Centre de gestion de la confidentialité inclut des paramètres de sécurité et de confidentialité. Ces paramètres vous aident à assurer la sécurité de votre ordinateur. Nous vous conseillons de ne pas les modifier.
Paramètres du Centre de gestion de la confidentialité...
OK
Annuler

Figure 16-1 – *Définissez les options de sécurité d'Excel dans cette fenêtre.*

– Emplacements approuvés. Vous trouverez ici la liste des dossiers approuvés. Les macros des documents qui s'y trouvent seront activées. Vous pouvez ajouter des dossiers à la liste par défaut.
– Documents approuvés. Les macros des documents approuvés sont activées.
– Compléments. Définissez dans cette fenêtre les options d'activation des compléments Excel – des modules qui ajoutent des fonctions aux programmes Office. Excel est livré avec un certain nombre de compléments, tels que le Solveur ou l'Assistant Somme conditionnelle. Pour connaître les compléments installés, en ajouter ou en supprimer, retournez à la fenêtre Options Excel et activez le volet Compléments.
– Paramètres ActiveX. Il s'agit des options d'activation des contrôles ActiveX.
– Paramètres des macros. Définissez ici les règles d'activation des macros :
  - Désactiver toutes les macros sans notification : les macros attachées aux documents qui ne se trouvent pas à un emplacement approuvé seront désactivées sans que l'utilisateur en soit averti.
  - Désactiver toutes les macros avec notification (valeur par défaut) : les macros des documents situés hors d'un emplacement approuvé seront désactivées et l'utilisateur en sera averti par un message (voir figure 16-2). Le bouton Activer le contenu (Options dans Excel 2007) active les macros du document pour la session en cours.

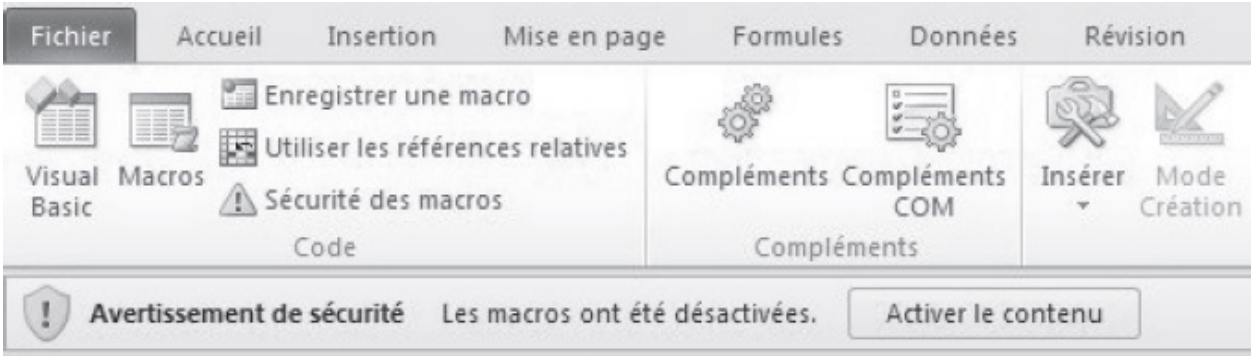


Figure 16-2 – *Par défaut, les macros sont désactivées et l'utilisateur en est averti via la barre de messages.*

  - Désactiver toutes les macros à l'exception des macros signées numériquement : celles situées à un emplacement autorisé ou signées

numériquement par un éditeur approuvé seront automatiquement activées. Dans le cas contraire, l'utilisateur pourra choisir de les activer ou non.

- Activer toutes les macros : elles seront toutes activées sans aucun contrôle du Centre de confidentialité. Pratique lorsque vous développez des projets VBA, cette option est déconseillée avec les documents provenant de sources extérieures.

– Mode protégé. Il ouvre les fichiers potentiellement dangereux en en désactivant les macros.

– Barre des messages. Zone où un potentiel message s'affiche lorsque le contenu actif d'un document ouvert est désactivé.

– Contenu externe. On définit ici le comportement d'Excel lorsque les macros établissent des connexions avec des sources externes, c'est-à-dire avec d'autres fichiers. Il peut s'agir de fichiers Excel ou d'autres types de données.

Astuce

Pour un accès direct à ces options, cliquez sur le bouton Sécurité des macros de l'onglet Développeur.

Les signatures numériques

Les signatures numériques permettent à leurs auteurs de certifier la provenance des macros qu'ils distribuent. Sélectionnez l'onglet Éditeurs approuvés du Centre de gestion de la confidentialité. Les macros portant les signatures listées ici sont considérées comme fiables. Elles sont donc activées sans que vous soyez prévenu de leur existence, quel que soit le niveau de sécurité choisi (figure 16-3).

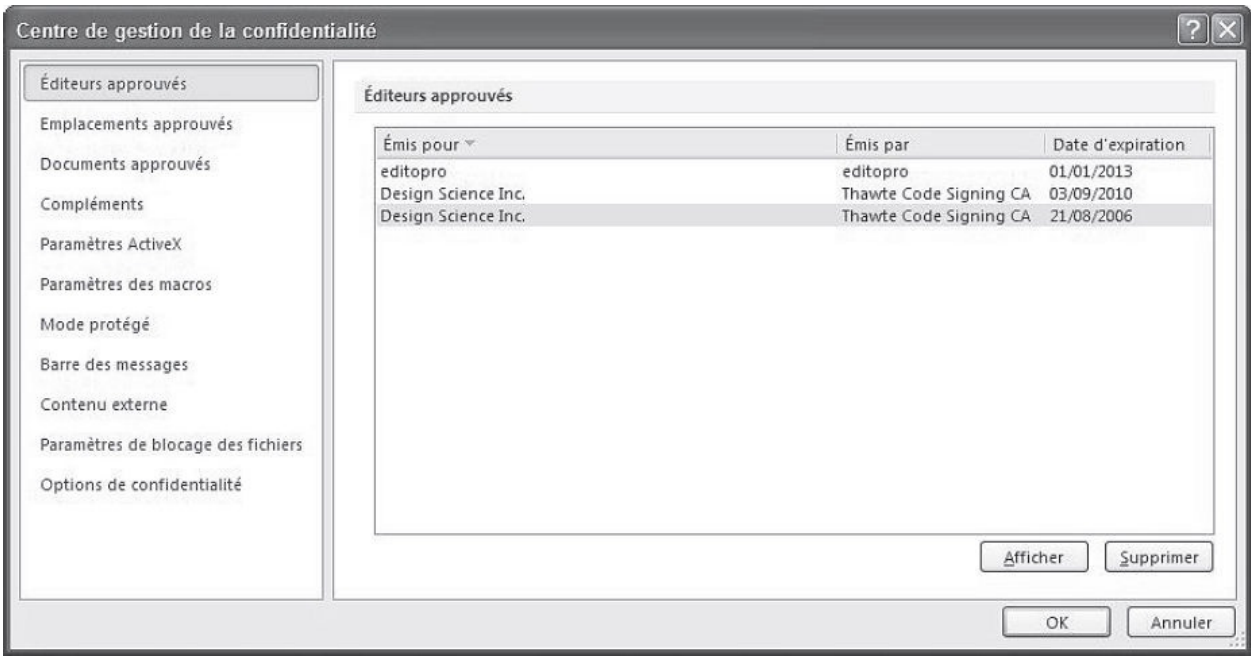


Figure 16-3 – *Les signatures numériques considérées comme sources fiables apparaissent ici.*

Conseil

Choisissez un niveau de sécurité adapté à vos utilisateurs et à votre usage des macros. Si vous travaillez dans une entreprise qui développe ses propres macros et n'utilise *a priori* pas de sources extérieures, optez pour la signature associée à un niveau de sécurité haut. C'est également le bon choix pour des utilisateurs qui ne sont pas conscients des dangers liés aux virus macros.

En revanche, si vous utilisez des macros provenant de sources extérieures variables, le niveau de sécurité élevé entraînerait la désactivation de toutes celles qui ne sont pas signées, sans que l'utilisateur ne soit prévenu de leur existence. Un niveau moyen se révélera alors plus adapté, à condition de sensibiliser les utilisateurs aux dangers des virus macros et de fixer des règles pour leur activation.

Le niveau de sécurité bas ne devrait être appliqué que dans des conditions exceptionnelles ; par exemple, si vous ne disposez pas d'une signature numérique et développez un nombre important de macros que vous souhaitez toujours activées.

Sauvegarder des macros

Vous pouvez prendre toutes les précautions imaginables, définir le niveau de sécurité le plus élevé et installer le tout dernier antivirus compatible Office, vous ne serez jamais à l'abri d'un virus qui contourne les barrières mises en place, ou d'un simple « crash » de votre disque dur ! Si vous avez passé des jours, voire des mois à développer des programmes VBA et si vous vous retrouvez du jour au lendemain sans la moindre ligne de code, vous risquez fort de détruire votre ordinateur.

Ne prenez pas de risques inutiles et respectez cette règle incontournable : sauvegardez régulièrement les classeurs et les modèles auxquels sont attachées vos macros (notamment PERSONAL.XLSB), ce qui conservera également les projets VBA correspondants. Une autre solution consiste à exporter les modules et les feuilles et à sauvegarder les fichiers ainsi générés en lieu sûr :

1. Sélectionnez le module ou la feuille dans l'Explorateur de projet.
2. Cliquez-droit et, dans le menu contextuel qui s'affiche, sélectionnez Exporter un fichier (voir figure 16-4).
3. Le type du fichier exporté varie selon l'élément supprimé :
   - Les modules standards ou modules de code sont exportés sous la forme de fichiers Basic portant l'extension .bas. Ces derniers sont des fichiers textes et peuvent donc être consultés et modifiés dans un éditeur tel que le Bloc-notes de Windows.
   - Les feuilles sont exportées sous la forme de deux fichiers Feuille portant les extensions .frm et .frx. Le premier est un fichier texte contenant quelques données propres à la feuille (telles que son nom, sa taille, sa position lors de l'affichage), ainsi que le code qui lui est attaché. Le second ne peut être lu dans un éditeur de texte et contient toutes les autres informations (essentiellement les données propres aux différents contrôles de la feuille).
   - Les modules de classes sont exportés sous la forme de fichiers Classe portant l'extension .cls.
4. Indiquez le répertoire et le nom d'enregistrement, puis cliquez sur le bouton Enregistrer.

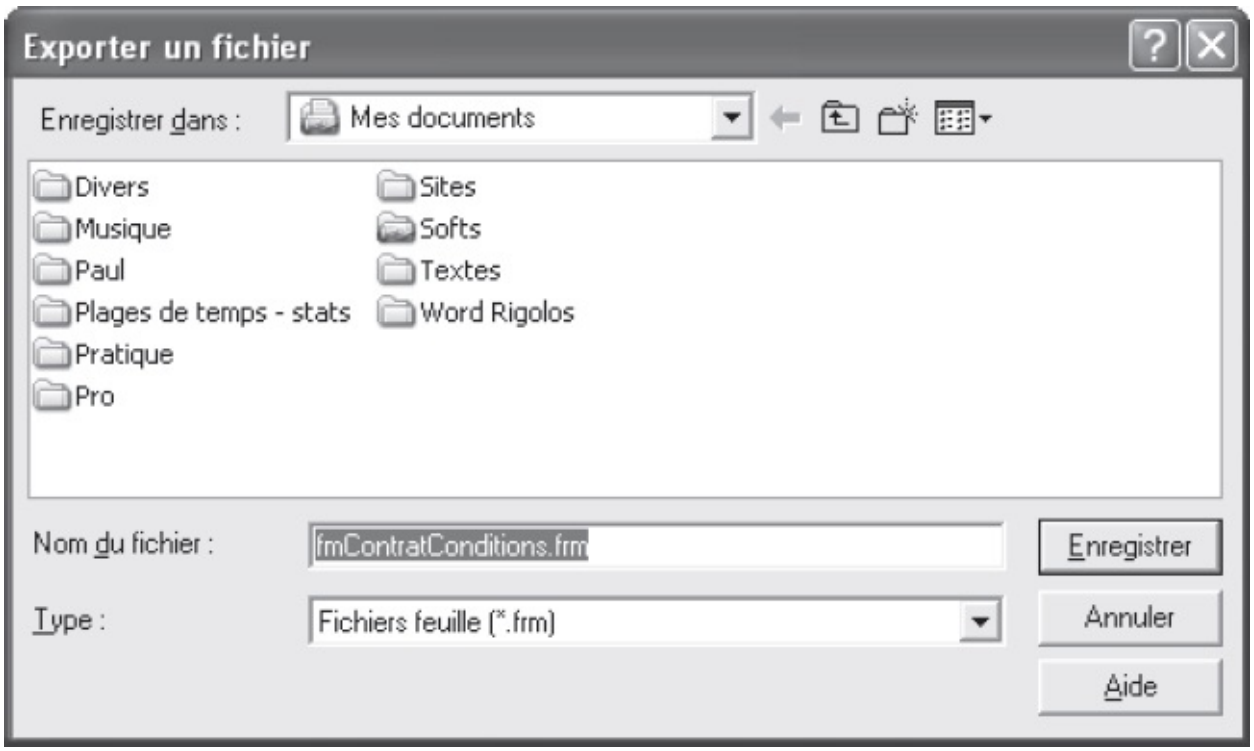


Figure 16-4 – *Indiquez le nom du fichier exporté et son dossier d'enregistrement.*

Le fichier sauvegardé est importable dans n'importe quel projet : affichez la fenêtre de l'Explorateur de projet, cliquez-droit sur l'un des éléments et choisissez la commande Importer un fichier. Sélectionnez ensuite le fichier (.bas, .frm ou .cls).

Protéger l'accès aux macros

Verrouiller un projet

Si d'autres utilisateurs accèdent à vos macros, un simple mot de passe empêchera qu'ils en voient/modifient le code. Pour verrouiller un projet, procédez comme suit :

1. Affichez l'Explorateur de projet.
2. Cliquez-droit sur le nom du projet et ouvrez ses propriétés.
3. Activez l'onglet Protection et cochez la case Verrouiller le projet pour l'affichage (voir figure 16-5).
4. Dans les zones définies à cet effet, saisissez à deux reprises le mot de passe

qui sera ensuite demandé pour accéder au projet, puis validez.

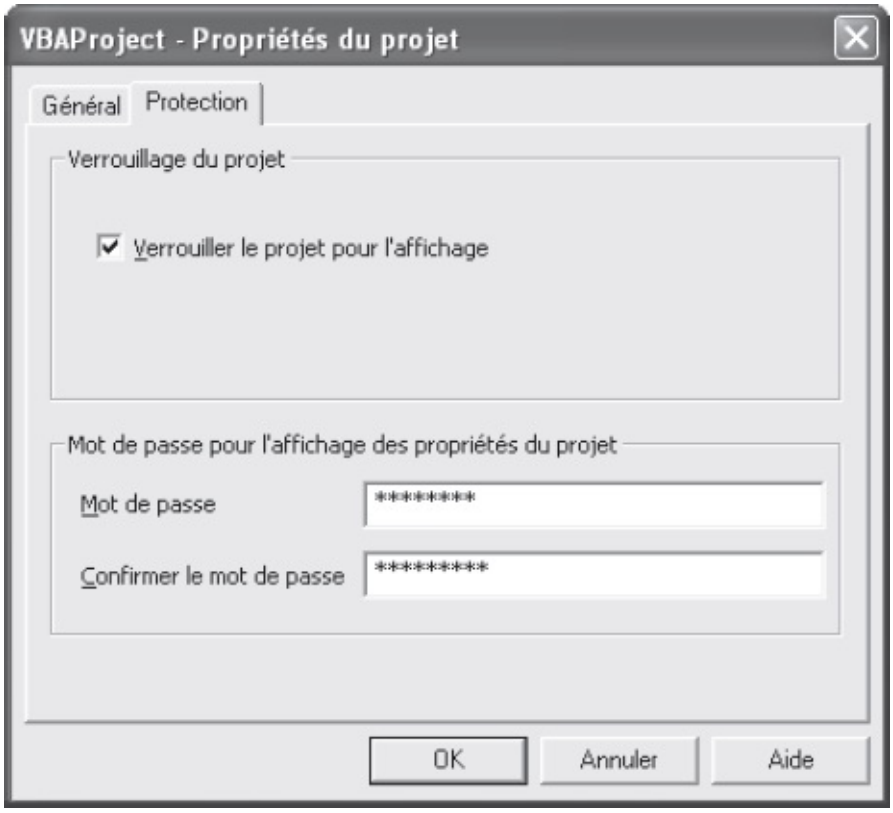


Figure 16-5 – *Protégez votre projet par un mot de passe.*

Le verrouillage du projet prendra effet dès la prochaine ouverture du document. Le mot de passe est nécessaire pour accéder au code du projet ou pour supprimer une macro (voir figure 16-6).

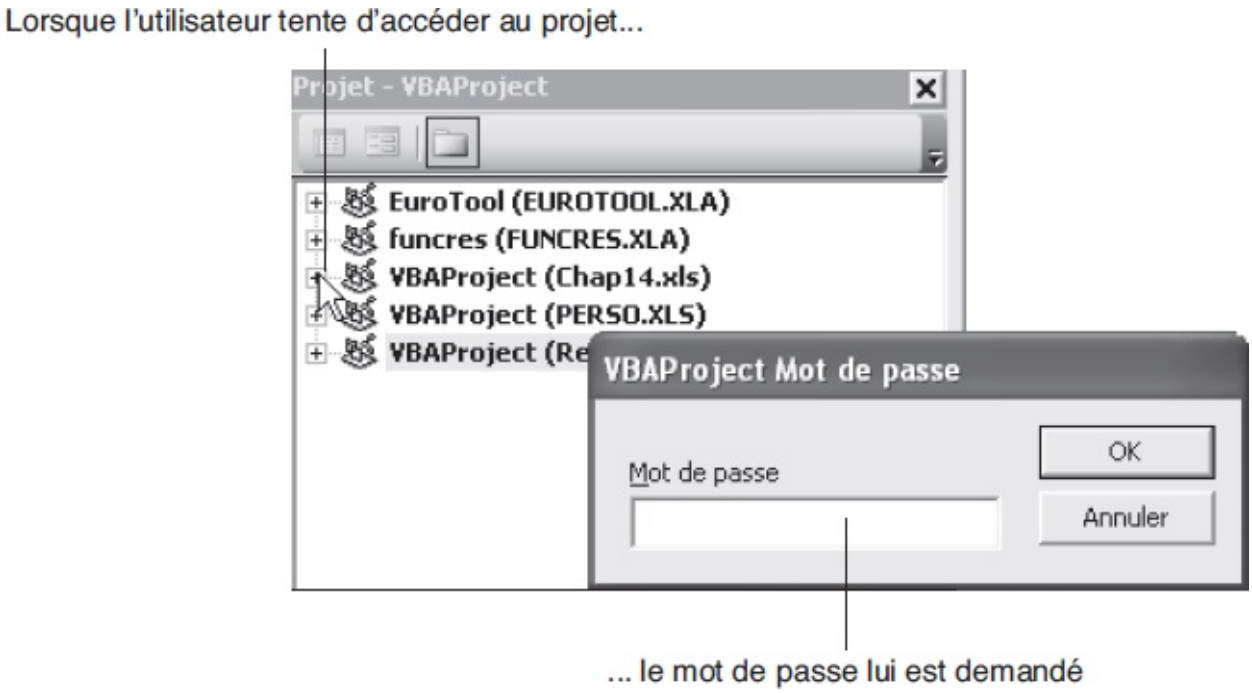


Figure 16-6 – *Le mot de passe est maintenant requis pour accéder au code du projet.*

Attention

Si vous oubliez le mot de passe défini au moment du verrouillage, vous n'aurez plus aucun moyen d'accéder au code. Il est fortement recommandé d'effectuer une sauvegarde du projet ou d'en exporter les modules et les feuilles avant de le protéger par mot de passe.

Limiter les droits d'exécution d'une macro

Il est parfois nécessaire de définir des autorisations pour l'exécution d'une macro, par exemple si elle donne accès à des informations confidentielles, ou encore quand elle peut causer des dégâts si elle n'est pas utilisée correctement.

La solution consiste alors à réclamer un mot de passe au moment de son exécution.

On crée pour cela une feuille intégrant une zone de texte dans laquelle l'utilisateur est invité à entrer le mot de passe. Une procédure vérifie que l'information fournie est correcte avant d'exécuter le code. Le projet devra évidemment être protégé contre l'affichage, afin que l'utilisateur ne puisse pas consulter le mot de passe dans le code de la feuille.

Voici une façon d'écrire le code de vérification du mot de passe. Créez une feuille UserForm nommée `fmMotDePasse` et placez-y un `Label`, un TextBox et deux

`CommandButton` (figure 16-7). Définissez-en les propriétés comme indiqué ci-après.

Propriété	Valeur
Feuille	
`Name`	`fmMotDePasse`
`Caption`	Entrez le mot de passe.
Contrôle *`Label`*	
`Name`	`lbMotdePasse`
`Caption`	Entrez ci-après le mot de passe requis pour l'exécution de cette commande.
Contrôle *`TextBox`*	
`Name`	`txtMotDePasse`
`PasswordChar`	*
Contrôle *`CommandButton 1`*	
`Name`	`cmdOK`
`Caption`	OK
`Default`	`True`
Contrôle *`CommandButton 2`*	
`Name`	`cmdAnnuler`
`Caption`	Annuler
`Cancel`	`True`

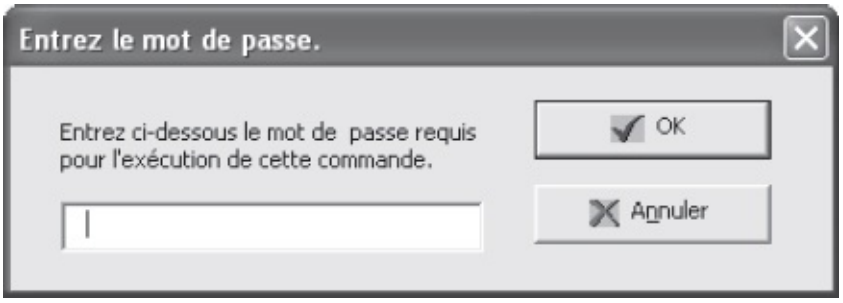


Figure 16-7 – *La feuille fmMotDePasse en mode Exécution.*

Insérez ensuite le code suivant dans la fenêtre Code de la feuille.

```
 1: Private Sub UserForm_Initialize()
 2:  txtMotDePasse.Value = ""
 3:  txtMotDePasse.SetFocus
 4: End Sub
 5:
 6: Private Sub cmdAnnuler_Click()
 7:  Unload Me
 8:  MsgBox "Cette commande ne peut être exécutée sans le mot de passe.", _
 9:    vbOKOnly + vbExclamation, "Fin de la commande"
10:  End
11: End Sub
12:
```

```
13: Private Sub cmdOK_Click()
14:  'On vérifie si le mot de passe entré est bon.
15:  If txtMotDePasse.Text="ftg87" Then
16:    Unload Me
17:  Else
18:    MsgBox "Le mot de passe fourni n'est pas correct.", _
19:      vbOKOnly + vbExclamation, "Mot de passe incorrect"
20:    txtMotDePasse.Value = ""
21:    txtMotDePasse.SetFocus
22:  End If
23: End Sub
```

Pour tester ce programme, insérez la procédure suivante dans n'importe quel module de code et exécutez-la.

```
Sub TestMotDePasse()
  fmMotDePasse.Show
  MsgBox "Si ce message s'affiche, c'est que le programme se poursuit."
End Sub
```

Cette procédure appelle la feuille de vérification du mot de passe (`fmMotDePasse`). Elle ne reprendra la main que si le mot de passe entré dans la feuille est correct ; elle affichera alors un message indiquant que le code se poursuit.

La première procédure (lignes 1 à 4) s'exécute à chaque affichage de la feuille. Elle affecte une chaîne vide à la zone de texte et lui passe le focus.

La procédure suivante (lignes 6 à 11) s'exécute lorsque l'utilisateur clique sur le bouton Annuler ou tape sur la touche Échap. La feuille est alors déchargée et un message informe l'utilisateur que la commande prend fin. L'instruction `End` met fin à l'exécution du programme.

La procédure `cmdOK_Click` se déclenche lorsque l'utilisateur clique sur le bouton OK ou tape sur la touche Entrée quand ce dernier a le focus. Lignes 15 à 22, une structure conditionnelle `If…Then…Else` vérifie que le mot de passe entré est correct, c'est-à-dire que l'utilisateur a saisi `ftg87` (ligne 15) dans la zone de texte. Si tel est le cas, la feuille est déchargée et la procédure ayant appelé l'affichage de la feuille reprend la main. Dans le cas contraire, un message indique à l'utilisateur que le mot de passe entré est incorrect. La zone de texte est alors vidée de la valeur précédemment entrée et reprend le focus.

Attention

Lignes 7 et 16, nous avons utilisé le mot-clé `Unload` et non `Hide` pour faire disparaître la feuille. Rappelez-vous que `Hide` masque la feuille mais ne libère pas les ressources mémoire qu'elle exploite. Si vous remplacez l'instruction `Unload Me` de la ligne 16 par l'instruction `Me.Hide`, la prochaine fois que la feuille sera affichée au cours de la session, le mot de passe précédemment saisi sera toujours dans la zone de texte. Il suffira alors à l'utilisateur de cliquer sur le bouton OK pour exécuter la macro.

Le programme présente ici un bogue grave : si l'utilisateur clique sur la croix

de fermeture de la fenêtre, aucun événement n'est détecté. La feuille se ferme et le programme se poursuit, comme si le mot de passe avait été fourni. Pour remédier à ce problème, ajoutez la procédure suivante au code de la feuille :

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
  If CloseMode=vbFormControlMenu Then
    Me.Hide
    MsgBox "Cette commande ne peut être exécutée sans le mot de passe.", _
      vbOKOnly + vbExclamation, "Fin de la commande"
    End
  End If
End Sub
```

L'événement `QueryClose` est détecté lorsqu'on tente de fermer la fenêtre. L'argument `Cancel` sert à annuler la fermeture. L'argument `CloseMode` indique la cause de l'événement ; ici, un clic sur le bouton de fermeture de la fenêtre (`vbFormControlMenu`). Si vous ne précisez pas `CloseMode`, la procédure s'exécutera dans n'importe quel contexte, y compris lorsque la méthode `Unload` sera utilisée dans le code. Dans ce cas, si l'utilisateur clique sur le bouton Annuler, le message l'informant que le programme prend fin sera affiché à deux reprises – dans le cadre des procédures événementielles `cmdAnnuler_Click` et `UserForm_QueryClose`.

Dans ce premier exemple, la feuille reste affichée jusqu'à ce que l'utilisateur entre le mot de passe correct ou qu'il clique sur le bouton Annuler. Vous pouvez également choisir de limiter le nombre d'essais. La procédure `cmdOK_Click` reproduite ci-après a été modifiée de sorte que le programme prenne fin après trois tentatives infructueuses de saisie du mot de passe. Nous avons également redéfini la propriété `Caption` de la feuille à "Entrez le mot de passe. Tentative 1 sur 3" et modifié le code de manière que cette information soit mise à jour dans la barre de titre de la fenêtre à chaque nouvelle tentative.

```
 1: Private Sub cmdOK_Click()
 2:   'La variable compteur servira à compter le nombre de tentatives.
 3:   Static compteur As Byte
 4:   compteur = compteur + 1
 5:
 6:   If txtMotDePasse.Text="ftg87" Then
 7:     Unload Me
 8:   Else
 9:
10:     'Si c'est la 3e fois que l'utilisateur entre un mot
11:     'de passe incorrect, le programme prend fin.
12:     If compteur=3 Then
13:       MsgBox "Echec dans la saisie du mot de passe." & _
14:         vbCr & "La commande ne peut être exécutée", _
15:         vbOKOnly + vbExclamation, "Mot de passe incorrect"
16:       End
17:     End If
18:
19:     MsgBox "Le mot de passe fourni n'est pas correct.", _
20:       vbOKOnly + vbExclamation, "Mot de passe incorrect"
```

```
21:     txtMotDePasse.Value = ""
22:     txtMotDePasse.SetFocus
23:     Me.Caption = "Entrez le mot de passe. Tentative " & _
24:       compteur+1 & " sur 3"
25:
26:   End If
27: End Sub
```

Exécutez la feuille. Lorsque vous saisissez un mot de passe incorrect, un message vous en informe et le titre de la fenêtre est mis à jour de façon à refléter le nombre de tentatives (voir figure 16-8). Si l'utilisateur saisit trois fois de suite un mot de passe erroné, un message l'informe que le programme ne peut être exécuté et la macro prend fin (voir figure 16-9).

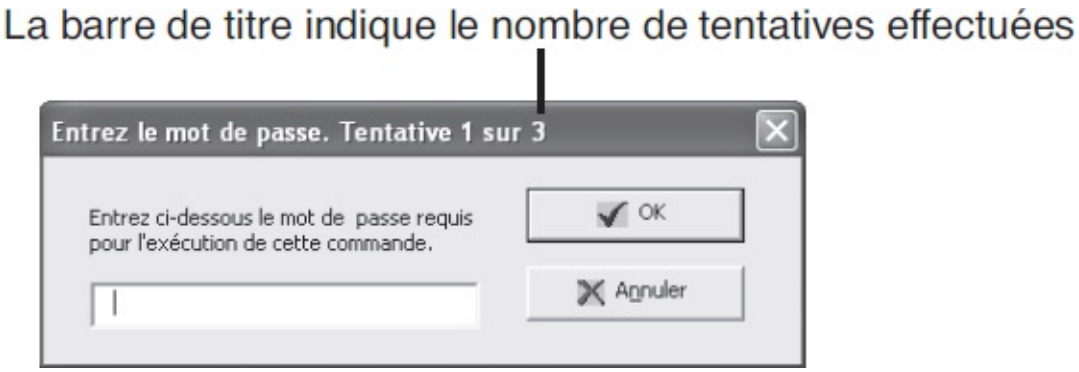

La barre de titre indique le nombre de tentatives effectuées
Entrez le mot de passe. Tentative 1 sur 3
Entrez ci-dessous le mot de passe requis pour l'exécution de cette commande.
OK
Annuler

Figure 16-8 – *Le nombre de tentatives s'affiche dans la barre de titre de la fenêtre.*

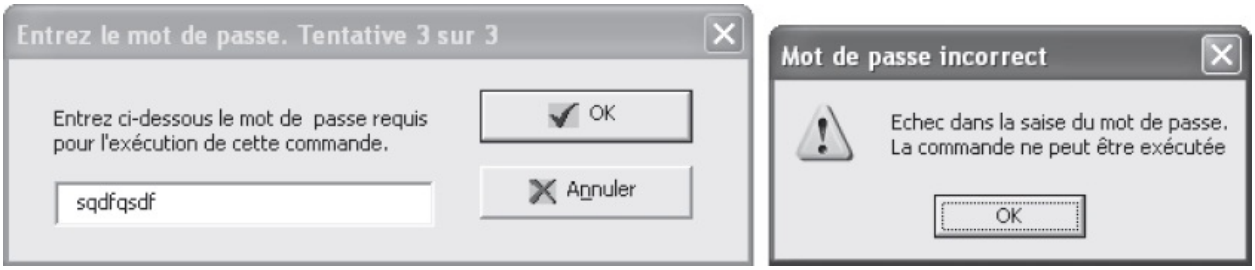

Entrez le mot de passe. Tentative 3 sur 3
Entrez ci-dessous le mot de passe requis pour l'exécution de cette commande.
sqdfqsdf
OK
Annuler
Mot de passe incorrect
Echec dans la saise du mot de passe. La commande ne peut être exécutée
OK

Figure 16-9 – *Après trois tentatives infructueuses, le programme prend fin.*

La variable `compteur` est déclarée à l'aide du mot-clé `Static` et est incrémentée de 1 à chaque exécution de la procédure, c'est-à-dire chaque fois que l'utilisateur valide un mot de passe. Notez que nous n'avons pas intégré d'instruction affectant la valeur 0 à `compteur` ; cela aurait en effet réinitialisé la variable à chaque exécution de la procédure, qui n'aurait alors jamais pris fin.

Rappel

Une variable statique conserve sa valeur entre les différents appels d'une procédure. Une variable

statique au sein d'un module de feuille conserve sa valeur tant que la feuille s'affiche.

Lignes 12 à 17, une structure `If…Then…Else` a été imbriquée dans la structure conditionnelle initiale. Elle vérifie si le compteur est égal à 3, c'est-à-dire si c'est la troisième fois que l'utilisateur propose un mot de passe erroné. Si tel est le cas, un message s'affiche et l'instruction `End` de la ligne 16 interrompt le programme.

Enfin, lignes 23 et 24, nous avons ajouté une instruction qui met à jour le texte affiché dans la barre de titre de la fenêtre. Pour définir quel sera le numéro de la prochaine tentative, on incrémente la valeur de Compteur (le nombre de tentatives effectuées, y compris celle en cours) de 1.

Telle qu'elle se présente dans les deux versions du programme que nous avons données ici, la feuille `fmMotDePasse` permet d'exiger un mot de passe de l'utilisateur à tout moment de l'exécution d'un programme. Il suffit pour cela de placer l'instruction :

```
fmMotDePasse.Show
```

Ce programme peut cependant être rendu plus souple et plus performant en contournant les limitations à un mot de passe (ici `ftg87`) et à trois tentatives, toutes deux définies dans le code de la feuille. Vous souhaiterez en effet probablement définir des mots de passe différents selon les programmes et les utilisateurs.

La solution consiste à appeler une procédure de validation en lui passant les valeurs « mot de passe » et « nombre de tentatives » comme arguments. Elle stockera ces valeurs dans des variables publiques de niveau module auxquelles les autres procédures de la feuille accéderont.

Créez un nouveau module de code et affectez-lui un nom représentatif, par exemple `MotDePasse`. Placez les instructions suivantes dans sa fenêtre de code :

```
1: Public varMotDePasse As String
2: Public varNumTentatives As Byte
3:
4: Sub ControleMotDePasse(MotDePasse As String, NumTentatives As Byte)
5:   varMotDePasse = MotDePasse
6:   varNumTentatives = NumTentatives
7:   fmMotDePasse.Show
8: End Sub
```

Lignes 1 et 2 les variables de niveau module `varMotDePasse` et `varNumTentatives` sont déclarées publiques. `ControleMotDePasse` (lignes 4 à 8) pourra être appelée par n'importe quelle procédure en lui passant les arguments requis, afin de subordonner la poursuite de l'exécution du programme en cours à la

présentation d'un mot de passe par l'utilisateur. Cette procédure affecte les valeurs qu'elle reçoit comme arguments aux variables `varMotDePasse` et `varNumTentatives` et appelle ensuite la feuille `fmMotDePasse`.

Modifiez ensuite le code de la feuille de la façon suivante :

```
 1: Private Sub UserForm_Initialize()
 2:   txtMotDePasse.Value = ""
 3:   txtMotDePasse.SetFocus
 4:   Me.Caption = "Entrez le mot de passe. Tentative 1" & _
 5:     " sur " & varNumTentatives
 6: End Sub
 7:
 8: Private Sub cmdAnnuler_Click()
 9:   Me.Hide
10:   MsgBox "Cette commande ne peut être exécutée sans le mot de passe.", _
11:     vbOKOnly + vbExclamation, "Fin de la commande"
12:   End
13: End Sub
14:
15: Private Sub cmdOK_Click()
16:   'La variable compteur servira à compter le nombre de tentatives.
17:   Static compteur As Byte
18:   compteur = compteur + 1
19:
20:   If txtMotDePasse.Text=varMotDePasse Then
21:     Unload Me
22:   Else
23:
24:     'Si l'utilisateur a utilisé toutes ses tentatives,
25:     'le programme prend fin.
26:     If compteur = varNumTentatives Then
27:       MsgBox "Echec dans la saise du mot de passe." & _
28:         vbCr & "La commande ne peut être exécutée", _
29:         vbOKOnly + vbExclamation, "Mot de passe incorrect"
30:       End
31:     End If
32:
33:     MsgBox "Le mot de passe fourni n'est pas correct.", _
34:       vbOKOnly + vbExclamation, "Mot de passe incorrect"
35:     txtMotDePasse.Value = ""
36:     txtMotDePasse.SetFocus
37:     Me.Caption = "Entrez le mot de passe. Tentative " & _
38:       compteur+1 & " sur " & varNumTentatives
39:
40:   End If
41: End Sub
44:
45: Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
46:   If CloseMode=vbFormControlMenu Then
47:     Me.Hide
48:     MsgBox "Cette commande ne peut être exécutée sans le mot de passe.", _
49:       vbOKOnly + vbExclamation, "Fin de la commande"
50:     End
51:   End If
52: End Sub
```

Partout dans le code où il était fait référence au nombre de tentatives ou au mot de passe, nous avons remplacé les valeurs fixes par des références aux variables `varMotDePasse` et `varNumTentatives`. Nous avons ajouté une instruction

définissant le titre de la fenêtre dans la procédure `Initialize` de la feuille (lignes 4 et 5), afin que le texte affiché dans la barre de titre reflète le nombre de tentatives autorisées dès l'affichage de la feuille. Ce titre est ensuite mis à jour à chaque passage de la fenêtre (ligne 38).

Ligne 20, la valeur de la zone de texte est maintenant comparée à celle stockée dans `varMotDePasse`, tandis que, ligne 26, `Compteur` l'est à `varNumTentatives`.

Pour imposer la saisie d'un mot de passe, il suffit maintenant de placer l'instruction suivante à l'endroit voulu :

```
Call ControleMotDePasse(MotDePasse, NumTentatives)
```

Pour tester le programme, placez la procédure suivante dans n'importe quel module du projet et exécutez-la (voir figure 16-10).

```
 1: Sub TestControleMotDePasse()
 2:   Dim maVar As String
 3:   Call ControleMotDePasse("Bonjour", 1)
 4:   maVar = Application.UserName
 5:   MsgBox "Mot de passe correct." & vbCr & "Poursuite de l'exécution.", _
 6:     vbOKOnly + vbInformation
 7:   Call ControleMotDePasse(maVar, 5)
 8:   maVar = Application.UserInitials & "-vba"
 9:   MsgBox "Mot de passe correct." & vbCr & "Poursuite de l'exécution.", _
10:   vbOKOnly + vbInformation
11:   Call ControleMotDePasse(maVar, 6)
12:   MsgBox "Mot de passe correct." & vbCr & "Poursuite de l'exécution.", _
13:   vbOKOnly + vbInformation
14: End Sub
```

L'instruction de la ligne 3 appelle le programme de contrôle et attend que le mot de passe « Bonjour » soit fourni dès la première tentative. Celle de la ligne 7 attend que le nom d'utilisateur enregistré pour l'application soit fourni et autorise 5 tentatives. Enfin, ligne 11, le mot de passe passé correspond aux initiales enregistrées pour l'utilisateur en cours, suivies de `-vba`, et autorise six tentatives. Chaque fois que la bonne information est fournie par l'utilisateur, `TestControleMotDePasse` reprend la main et un message s'affiche (lignes 5, 9 et 12).

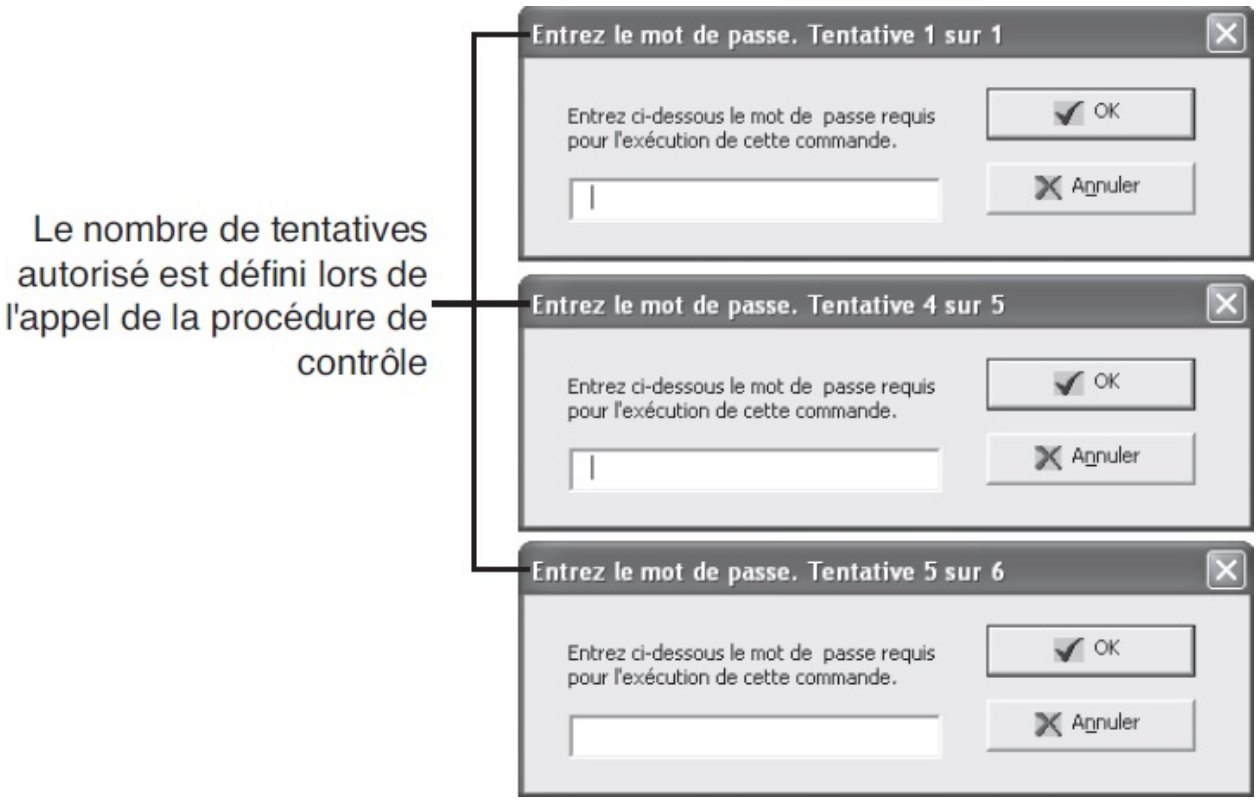

Le nombre de tentatives autorisé est défini lors de l'appel de la procédure de contrôle
Entrez le mot de passe. Tentative 1 sur 1
Entrez ci-dessous le mot de passe requis pour l'exécution de cette commande.
OK
Annuler
Entrez le mot de passe. Tentative 4 sur 5
Entrez ci-dessous le mot de passe requis pour l'exécution de cette commande.
OK
Annuler
Entrez le mot de passe. Tentative 5 sur 6
Entrez ci-dessous le mot de passe requis pour l'exécution de cette commande.
OK
Annuler

Figure 16-10 – *Le titre de la fenêtre reflète le nombre d'essais autorisés.*

Astuce

Vous pouvez également limiter le nombre de caractères autorisés dans la zone de texte à la longueur du mot de passe attendu. Insérez pour cela l'instruction suivante dans la procédure `UserForm_Initialize` de `fmMotDePasse` :

```
txtMotDePasse.MaxLength = Len(varMotDePasse).
```

Rappel

La propriété `MaxLength` est développée au chapitres 14.

Apportons une ultime amélioration au programme : autorisons plusieurs mots de passe pour exécuter une macro. Cela sera nécessaire si vous définissez un mot de passe par « groupe d'utilisateurs ». Vous limiterez par exemple l'accès à certaines macros aux utilisateurs du groupe « Financier » et d'autres au groupe « Production ».

Voici les modifications à apporter au programme pour autoriser jusqu'à trois mots de passe (module `MotDePasse`) :

```
1: Public varMotDePasse1 As String, varMotDePasse2 As Variant,
  ➥varMotDePasse3 As Variant
2: Public varNumTentatives As Byte
```

```
 3:
 4: Sub ControleMotDePasse(NumTentatives As Byte, MotDePasse1 As String,
    ➥ Optional MotDePasse2 As String, Optional MotDePasse3 As String)
 5:   varNumTentatives = NumTentatives
 6:   varMotDePasse1 = MotDePasse1
 7:
 8:   'On vérifie si plusieurs mots de passe ont été passés,
 9:   'et on affecte une valeur aux variables en conséquence.
10:   If IsMissing(MotDePasse2)=True Then
11:     varMotDePasse2 = MotDePasse1
12:   Else
13:     varMotDePasse2 = MotDePasse2
14:   End If
15:   If IsMissing(MotDePasse3)=True Then
16:     varMotDePasse3 = MotDePasse1
17:   Else
18:     varMotDePasse3 = MotDePasse3
19:   End If
20:   fmMotDePasse.Show
21: End Sub
```

Ligne 1, nous avons créé trois variables destinées à recevoir les 3 mots de passe autorisés. Nous avons ensuite modifié la déclaration de `ControleMotDePasse` (ligne 4) en y ajoutant les arguments optionnels `MotDePasse2` et `MotDePasse3`. Notez que nous avons dû déplacer l'argument `Numtentatives` en début de liste car un argument optionnel ne peut être suivi d'un obligatoire.

Deux structures conditionnelles (lignes 10 à 14 et lignes 15 à 19) vérifient ensuite si les mots de passe optionnels ont été passés. Si l'argument correspondant au deuxième mot de passe valide a été passé, sa valeur est affectée à `varMotDePasse2` ; sinon `varMotDePasse2` vaut `MotDePasse1`. Le même traitement est appliqué à `varMotDePasse3`.

Rappel

Le mot-clé `Optional` indique qu'un argument n'est pas obligatoire. La fonction `IsMissing` sert pour vérifier si l'argument a été passé ou non.

Attention

La fonction `IsMissing` ne fonctionne correctement qu'avec des arguments de type `Variant`. Si vous déclarez les arguments `MotDePasse2` et `MotDePasse3` de type `String`, `IsMissing` renverra toujours `False`. En conséquence, `varMotDePasse2` et `varMotDePasse3` se verront affecter une chaîne vide, qui sera reconnue comme mot de passe valide.

Modifiez ensuite le test de la procédure `cmdOK_Click` de façon que chacun des trois mots de passe autorisés soit reconnu comme valide :

```
If txtMotDePasse.Text=varMotDePasse1 Or txtMotDePasse.Text=varMotDePasse2
➥Or txtMotDePasse.Text=varMotDePasse3 Then
```

Info

Grâce aux instructions des lignes 10 à 19, `varMotDePasse2` et `varMotDePasse3` ne sont jamais vides, même si les arguments `MotDePasse2` et/ou `MotDePasse3` n'ont pas été passés à la procédure. Ce traitement est nécessaire pour qu'aucune erreur ne soit provoquée lors de la vérification du texte saisi par l'utilisateur.

Pour appeler le programme mot de passe, il suffit maintenant de placer l'instruction suivante à l'endroit voulu :

```
Call ControleMotDePasse(NumTentatives, MotDePasse1, MotDePasse2, MotDePasse3)
```

Pour le tester, placez la procédure suivante dans n'importe quel module du projet et exécutez-la.

```
 1: Sub TestControlePlusieursMotsDePasse()
 2:   Call ControleMotDePasse(5, "Bonjour")
 3:   MsgBox "Mot de passe correct." & vbCr & "Poursuite de l'exécution.", _
 4:     vbOKOnly + vbInformation
 5:   Call ControleMotDePasse(5, "Bonjour", "Salut")
 6:   MsgBox "Mot de passe correct." & vbCr & "Poursuite de l'exécution.", _
 7:     vbOKOnly + vbInformation
 8:   Call ControleMotDePasse(5, "Bonjour", "Salut", "Coucou")
 9:   MsgBox "Mot de passe correct." & vbCr & "Poursuite de l'exécution.", _
10:     vbOKOnly + vbInformation
11: End Sub
```

Chacune des instructions appelant la procédure de vérification du mot de passe (lignes 2, 5 et 8) autorise 5 tentatives. Lors du premier appel, seul le mot de passe "Bonjour" est passé comme argument. Le deuxième appel fournit aussi la valeur "Salut" pour l'argument optionnel `MotDePasse2`. L'un ou l'autre des mots de passe sera donc accepté. Enfin, lors du troisième appel, tous les arguments sont passés à la procédure `ControleMotDePasse` et les mots de passe "Bonjour", "Salut" et "Coucou" seront tous trois acceptés. Chaque fois que le bon mot de passe est fourni par l'utilisateur, la procédure `TestControlePlusieursMotsDePasse` reprend la main et un message s'affiche (lignes 3 et 9).

Authentifier ses macros

Les *signatures numériques* servent à authentifier vos macros en identifiant leur source (le nom de la personne ou de la société les ayant développées). À l'ouverture d'un document, le détail des signatures des éventuelles macros s'affiche à l'écran.

Conseil

Le fait que des macros soient numériquement signées ne vous garantit pas leur fiabilité. Un développeur

malhonnête peut se procurer une signature électronique pour signer des virus macros. Ne vous fiez aux signatures électroniques que si elles authentifient une personne ou une société de votre connaissance.

Obtenir une authentification

Pour authentifier vos macros, vous devez vous procurer une signature électronique auprès du service approprié. Un certificat authentifié peut être acheté auprès de sociétés reconnues, telles VeriSign (**http://www.verisign.com**) ou Thawte (**http://www.thawte.com**).

Vous avez également la possibilité de créer vos propres certificats numériques (non authentifiés) en utilisant l'utilitaire fourni avec Microsoft Office. Ouvrez le menu Démarrer puis choisissez Outils Microsoft Office > Certificat numérique pour les projets VBA (voir figure 16-11).

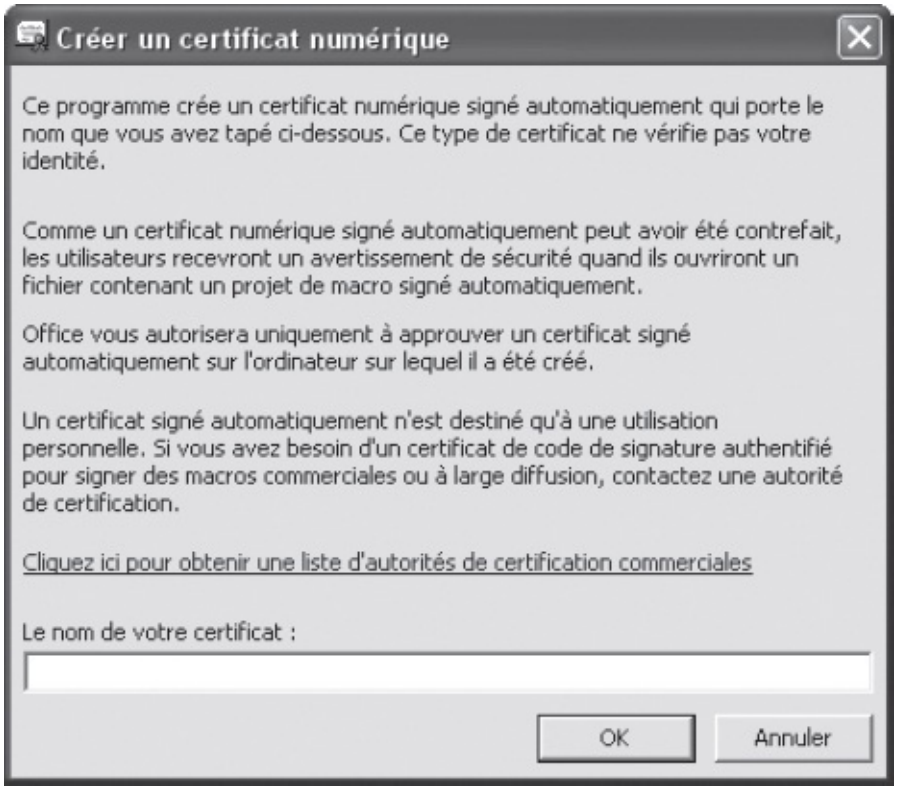

Créer un certificat numérique
Ce programme crée un certificat numérique signé automatiquement qui porte le nom que vous avez tapé ci-dessous. Ce type de certificat ne vérifie pas votre identité.
Comme un certificat numérique signé automatiquement peut avoir été contrefait, les utilisateurs recevront un avertissement de sécurité quand ils ouvriront un fichier contenant un projet de macro signé automatiquement.
Office vous autorisera uniquement à approuver un certificat signé automatiquement sur l'ordinateur sur lequel il a été créé.
Un certificat signé automatiquement n'est destiné qu'à une utilisation personnelle. Si vous avez besoin d'un certificat de code de signature authentifié pour signer des macros commerciales ou à large diffusion, contactez une autorité de certification.
Cliquez ici pour obtenir une liste d'autorités de certification commerciales
Le nom de votre certificat :
OK
Annuler

Figure 16-11 – *Vous pouvez créer vos propres certificats numériques.*

Info

Si la commande Certificat numérique n'apparaît pas dans le groupe Microsoft Office, recherchez le fichier Selfcert.exe.

Info

L'outil selfcert n'est pas installé par défaut avec Office 365.

Pour une liste exhaustive des organismes de certification, consultez l'aide en ligne d'Office ou, dans la boîte de dialogue représentée à la figure 16-11, cliquez sur le lien.

Authentifier une macro

Lorsque vous avez obtenu une signature électronique, il ne vous reste qu'à « signer » vos macros :

1. Sélectionnez le projet dans l'Explorateur de projet.
2. Choisissez Outils > Signature électronique (voir figure 16-12).
3. Cliquez sur le bouton Choisir et, dans la fenêtre qui s'affiche, choisissez la signature électronique que vous souhaitez affecter à votre projet.

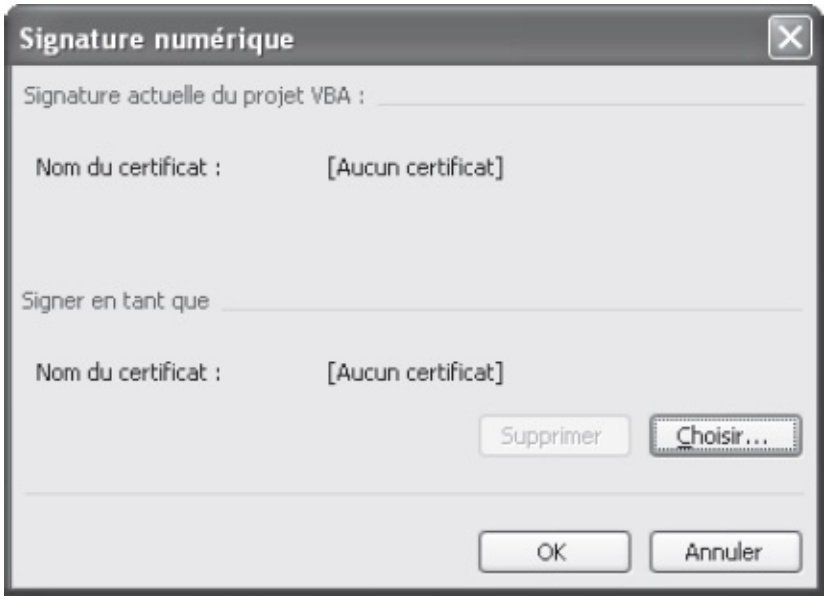


Figure 16-12 – *La boîte de dialogue Signature numérique gère les signatures affectées à vos projets.*

17

Exemple complet d’application Excel

Ce chapitre constitue un récapitulatif. Nous vous proposons d’y créer un programme complet, étape par étape, de sa définition à son intégration dans l’interface d’Excel, qui mette en pratique l’ensemble des connaissances acquises au cours de cet ouvrage.

Info

Si ce n’est déjà fait, téléchargez les codes sources des exemples du livre à l’adresse suivante : http://www.editions-eyrolles.com/dl/0067401. Commencez par décompresser l’archive, puis testez le programme que nous développerons dans ce chapitre en ouvrant le fichier chap17.xlsx du dossier Bonus.

Présentation du projet d’application Excel

Le programme que nous allons écrire ici aura pour fonction de générer des factures de droits d’auteur et de mettre à jour un tableau Word. La figure 17-1 présente une facture type.

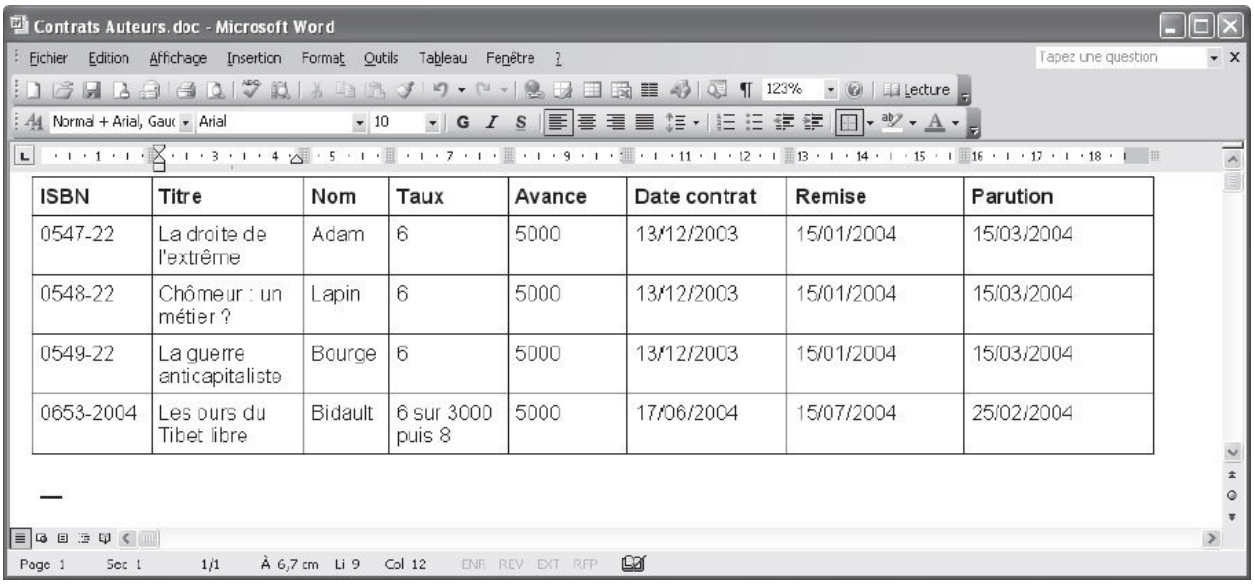
Afin de simplifier et de sécuriser cette tâche, nous créerons des interfaces utilisateur adaptées, dans lesquelles il suffira d’entrer les données nécessaires au programme. Ce dernier composera les feuilles de paie, les enregistrera et en imprimera un nombre d’exemplaires défini par l’utilisateur. Il mettra également à jour un tableau dans un document Word répertoriant les informations essentielles concernant le contrat.

	A	B	C	D	E	F	G
1	**DROITS D'AUTEURS**						
2	**AUTEUR / SOCIETE**		Mikaël Bidault				
3	**TITRE**		Les ours du tibet				
4	**ISBN**		111-71				
5							
6	**DATE DE REMISE**		25/12/2004				
7	**DATE DE PARUTION**		01/02/2005				
8							
9	**ADRESSE**		**5 rue du soleil**				
10	**CODE POSTAL**		**75020**				
11	**VILLE**		**Paris**				
12	**PAYS**		France				
13	**DROITS D'AUTEUR**						
14							
15	**MONTANT BRUT A PAYER**					**2 500,00**	
16	TVA (0,8%)					20,00	
17	AGESSA (0,85%)					21,25	
18	BASE CSG (95% MONTANT BRUT)					2 375,00	
19	CSG (7,50%)					178,13	
20	BASE RDS (95% MONTANT BRUT)					2 375,00	
21	RDS (0,5%)					11,88	
22							
23	**NET A PAYER**					**2 308,75**	
24							
25							
26	TVA A VERSER					117,50	
27	CONTRIBUTION DIFFUSEUR (1%)					25,00	
28							
29							
30	**DATE DE REGLEMENT**						
31							
32	**DROITS D'AUTEUR**	4000 premiers exemplaires				8%	(huit pour cent)
33		Au-delà				10%	(dix pour cent)
34							

Figure 17-1 – *Une facture de droits d'auteur type.*

Avant de commencer

Pour les besoins du programme, créez un fichier Word semblable à celui de la figure 17-2 et enregistrez-le sous le nom Contrats Auteurs.doc dans le dossier C:\Mes documents\.


Contrats Auteurs.doc - Microsoft Word
Fichier Edition Affichage Insertion Format Outils Tableau Fenêtre ?
Tapez une question
Lecture
Normal + Arial, Gauc
Arial
10
Page 1 Sec 1 1/1 À 6,7 cm Li 9 Col 12

ISBN	Titre	Nom	Taux	Avance	Date contrat	Remise	Parution
0547-22	La droite de l'extrême	Adam	6	5000	13/12/2003	15/01/2004	15/03/2004
0548-22	Chômeur : un métier ?	Lapin	6	5000	13/12/2003	15/01/2004	15/03/2004
0549-22	La guerre anticapitaliste	Bourge	6	5000	13/12/2003	15/01/2004	15/03/2004
0653-2004	Les ours du Tibet libre	Bidault	6 sur 3000 puis 8	5000	17/06/2004	15/07/2004	25/02/2004

Figure 17-2 – *Chaque fois qu'un contrat est édité, les données sont intégrées dans un fichier Word.*

Le programme VBA que vous développerez dans ce chapitre utilise le contrôle Calendrier pour inviter l'utilisateur à sélectionner des dates. Celui-ci n'est pas activé par défaut, commencez par le référencer dans la boîte à outils de Visual Basic. La procédure est décrite à la section « Personnaliser la boîte à outils » du chapitres 12.

Attention

Si vous installez vos macros sur différents ordinateurs, les contrôles personnalisés devront également être dupliqués.

Une fois le contrôle Calendrier installé sur votre ordinateur, testez le programme. Ouvrez le fichier chap17.xls des codes sources du livre. Si Excel affiche un message d'avertissement, activez le contenu du fichier, puis cliquez sur le bouton Editer un contrat. Suivez les étapes du programme. Jetez maintenant un œil au dossier contenant le fichier chap17.xls : il contient les feuilles de paie que le programme a éditées. Par ailleurs, le document Contrat auteurs.doc a été mis à jour.

Identification des informations à recueillir

Les informations nécessaires à l'établissement d'une feuille de paie de droits d'auteur sont les suivantes :

- Auteur. Nom et prénom (ou société), adresse, code postal, ville, pays (facultatif).

  Les adresses des auteurs sont stockées dans un fichier Excel (voir figure 17-3). Une interface permettra de sélectionner l'auteur dans la liste et mettra à jour son adresse sans qu'il soit nécessaire pour l'utilisateur de la connaître. Si l'auteur ne fait pas partie de la liste, l'utilisateur pourra saisir l'ensemble de ses coordonnées et les ajouter au fichier Excel.

  Le contrat sera établi au nom d'une personne physique ou à celui d'une société. Dans le premier cas, un prénom devra être fourni.

Info

Pour les besoins du programme, créez un fichier Excel semblable à celui de la figure 17-3 et enregistrez-le sous le nom Classeur Auteurs.xlsx dans le dossier C:\Mes documents.

- Ouvrage. Titre de l'ouvrage et ISBN (*International Standard Book Number*, numéro identifiant l'ouvrage de façon unique).

  L'ISBN sera toujours formaté de la façon suivante : xxxx-y, où chaque x correspond à un chiffre et chaque y à une clé pouvant être un chiffre ou une lettre.

- Conditions de rémunération. Droits d'auteur et avance sur droits d'auteur.

  Les droits d'auteur sont un pourcentage du prix de vente de l'ouvrage, soit fixe, soit variable (par exemple, 6 % sur les 3 000 premiers livres, puis 8 %).

  L'avance sur droits d'auteur est facultative. Dans le cas d'une avance, celle-ci pourra être versée en une fois (à la remise du manuscrit) ou en deux fois (une moitié à la remise et une moitié à la parution).

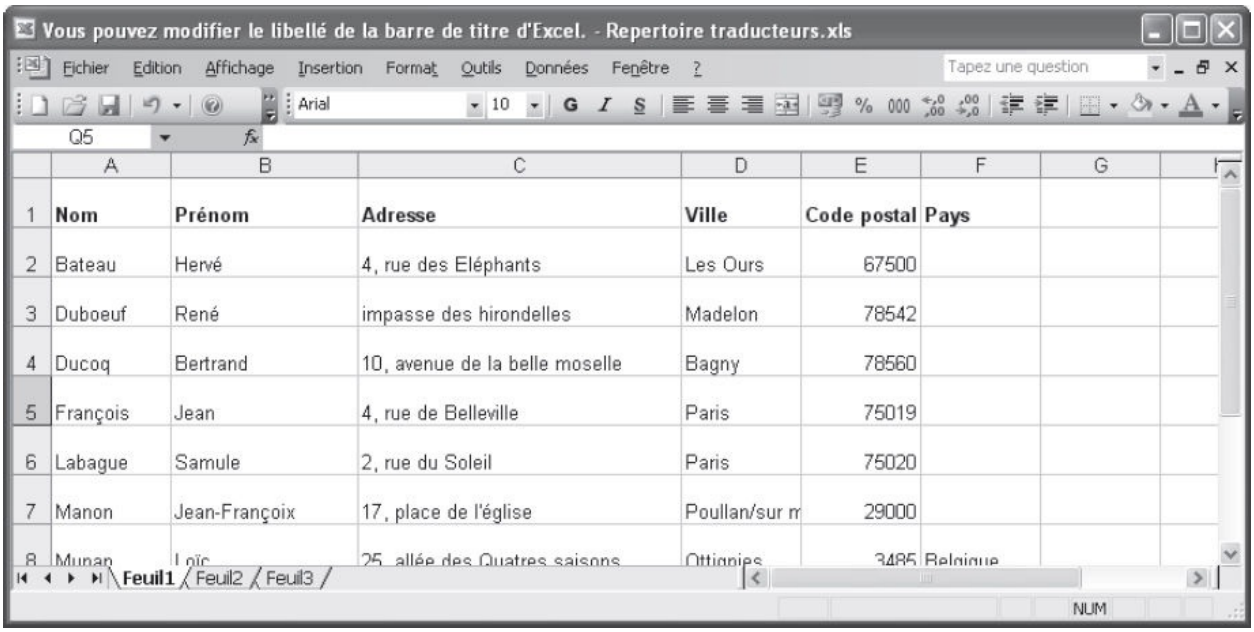

Vous pouvez modifier le libellé de la barre de titre d'Excel. - Repertoire traducteurs.xls
Fichier Edition Affichage Insertion Format Outils Données Fenêtre ?
Tapez une question
Arial 10
Q5
Nom Prénom Adresse Ville Code postal Pays
Bateau Hervé 4, rue des Eléphants Les Ours 67500
Duboeuf René impasse des hirondelles Madelon 78542
Ducoq Bertrand 10, avenue de la belle moselle Bagny 78560
François Jean 4, rue de Belleville Paris 75019
Labague Samule 2, rue du Soleil Paris 75020
Manon Jean-Françoix 17, place de l'église Poullan/sur m 29000
Feuil1 Feuil2 Feuil3
NUM

Figure 17-3 – *La liste des auteurs est stockée dans un fichier Excel.*

- Taxes. Les taxes retenues sur l'avance sur droits d'auteur sont la TVA, les AGESSA, la CSG et le RDS. Le programme n'aura cependant pas à les traiter, puisque nous créerons un classeur modèle dans lequel nous intégrerons les formules nécessaires.
- Dates. Date de remise et date de parution.

  La date de parution devra toujours être postérieure à la date de remise. Par ailleurs, si moins de 40 jours séparent la date de remise de la date de parution, une confirmation sera demandée à l'utilisateur.
- Options d'impression. L'utilisateur aura le choix d'imprimer ou non les documents édités. Il pourra également définir le nombre d'exemplaires imprimés pour le courrier et pour les feuilles de paie.

Définition de la structure du programme

Pour assurer une bonne lisibilité au programme et être en mesure de réutiliser une partie du code dans d'autres conditions (par exemple, pour un programme équivalent concernant un travail de traduction), les procédures seront aussi autonomes qu'il se peut. Nous devons ici définir le squelette du programme (son flux), les modules de stockage des procédures et la façon dont seront stockées les informations fournies par l'utilisateur.

Le squelette du programme

Le programme sera structuré de la façon suivante :

1. Déclaration des variables objets et liaison avec les fichiers à lire ou à manipuler (répertoire des auteurs et document Word à compléter).
2. Affichage des interfaces utilisateur.

   On commence par se procurer l'ensemble des informations requises. Pour chaque interface :

   a. Chargement des feuilles (procédures `Initialize`).

   Les interfaces sont appelées à partir de la procédure principale.

   b. Vérification des informations fournies par l'utilisateur.

   Les procédures événementielles des feuilles vérifient la validité des données.

   c. Stockage des données dans des variables.

   Lorsque les données sont valides, elles sont affectées aux variables appropriées, pour être ensuite exploitées par le programme.

   d. Masquage de l'interface.
3. Édition, impression éventuelle et enregistrement des feuilles de paie.

   Un nouveau classeur fondé sur un modèle est créé. Les informations nécessaires sont entrées dans les cellules appropriées.
4. Mise à jour du tableau Word.

   Une ligne y est ajoutée et les données du contrat y sont intégrées.
5. Fin du programme et libération des ressources mémoire.

Créez un nouveau module de code et appelez-le `ContratAuteur`. Placez-y tout de suite le corps de la procédure principale, en commentant les phases principales :

```
Sub EditionContratAuteur()
'1. Liaisons des variables objets
'2. Affichage des feuilles
'3. Edition des feuilles de paie
'4. Mise à jour du tableau Word
'5. Libération des ressources mémoire
End Sub
```

Les modules

Créez les modules suivants pour accueillir les procédures :

- ContratAuteur. On y retrouvera les déclarations de variables, la procédure principale et les procédures spécifiques au contrat d'auteur. Les variables devront être publiques pour être manipulées à partir de n'importe quel module, notamment à partir des interfaces afin de leur affecter les valeurs entrées par l'utilisateur.
- ContratsDivers. Ce module regroupera les procédures susceptibles d'être réutilisées par d'autres programmes (par exemple un programme de contrat de traduction), telles que l'impression, l'enregistrement, etc.
- RepertoireAuteurs. Ce module regroupera les procédures destinées à manipuler le répertoire des auteurs. Elles devront être publiques pour être appelées à partir du module `ContratAuteur` et manipulées à partir des interfaces utilisateur.

Le dossier Modules de l'Explorateur de projet doit être semblable à celui de la figure 17-4.

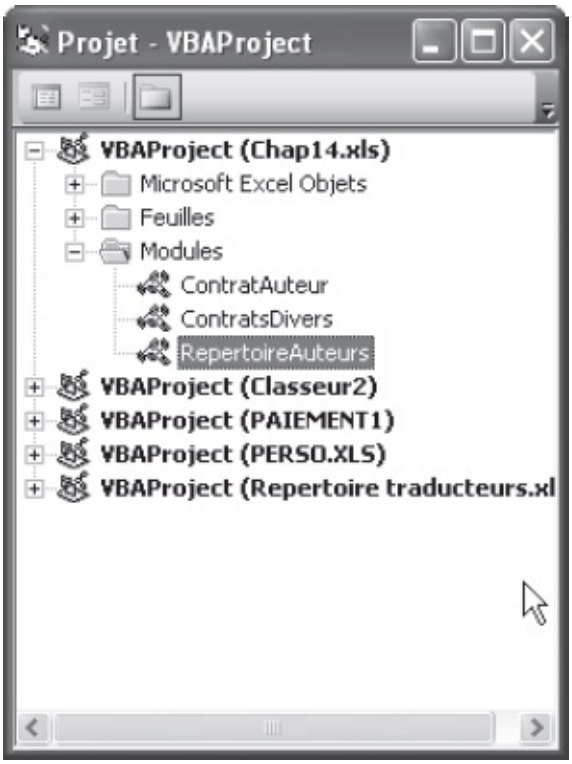


Figure 17-4 – *L'Explorateur de projet à ce stade de la création du programme.*

Le stockage des informations fournies par l'utilisateur

Les valeurs fournies par l'utilisateur au travers des interfaces qu'il sera invité à compléter seront stockées dans des types de données personnalisés

regroupant des informations liées :

- Le type `Auteur` contiendra : nom (`String`), prénom (`String`), adresse (`String`), code postal (`String`), ville (`String`), pays (`String`), société (`Boolean` : société = `True`, personne physique = `False`).
- Le type `ConditionsAuteur` rassemblera les données relatives à l'ouvrage et aux droits d'auteur : titre (`String`), ISBN (`String`), taux variable ou non (`Boolean`), taux initial des droits d'auteur (`String` et `Byte`), nombre d'exemplaires affectés par le taux initial (`Integer`), taux au-delà de x exemplaires (`String` et `Byte`), avance ou non (`Boolean`), valeur de l'avance (`Integer`), nombre de paiements (`Byte`), date de remise (`Date`) et date de parution (`Date`).
- Le type `OptionsImpression` rassemblera : imprimer ou non (`Boolean`), imprimer ou non le courrier (`Boolean`), imprimer ou non les feuilles de paie (`Boolean`), nombre d'exemplaires du courrier à imprimer (`Variant`) et nombre d'exemplaires des feuilles de paie à imprimer (`Variant`).

Placez d'ores et déjà les déclarations de ces types en tête du module `ContratAuteur` :

```
'Types de données personnalisées

Type Auteur
    Prenom As String
    Nom As String
    Adresse As String
    CodePostal As String
    Ville As String
    Pays As String
    Societe As Boolean
End Type

Type ConditionsAuteur
    Titre As String
    ISBN As String
    TauxVariable As Boolean
    TauxDroitsNum1 As Byte
    TauxDroitsNum2 As Byte
    TauxDroitsStr1 As String
    TauxDroitsStr2 As String
    NumExemplairesTaux1 As Integer
    AvanceOuNon As Boolean
    AvanceSurDroits As Integer
    NumPaiements As Byte
    remise As Date
    Parution As Date
End Type

Type OptionsImpression
    Imprimer As Boolean
    ImprimerCourrier As Boolean
    ImprimerPaie As Boolean
    nbreCourrier As Variant
```

```
    nbrePaie As Variant
End Type
```

Ajoutez en-dessous la déclaration des variables publiques dans lesquelles seront stockées les données. Elles doivent se trouver dans la zone de déclaration du module (hors de toute procédure) pour être accessibles dans tout le reste du projet.

```
Public MonAuteur As Auteur
Public MesConditionsAuteur As ConditionsAuteur
Public MonImpression As OptionsImpression
Public ClasseurAuteurs As Workbook
Public MonTableauWord
```

Les trois premières instructions déclarent les variables dans lesquelles seront stockées les informations fournies par l'utilisateur. Les deux dernières variables se verront respectivement affecter le classeur des auteurs et le document Word contenant le tableau récapitulatif des contrats.

La figure 17-5 présente le module `ContratAuteur` tel qu'il doit se présenter à ce stade.

Attention

Avant de manipuler des fichiers Word, vous devez créer une référence à la bibliothèque d'objets de l'éditeur de texte. Choisissez la commande Références du menu Outils de Visual Basic Editor et cochez la case Microsoft Word Object Library (voir figure 17-6). Si vous omettez de le faire, l'instruction de déclaration de la variable `MonTableauWord` générera une erreur.

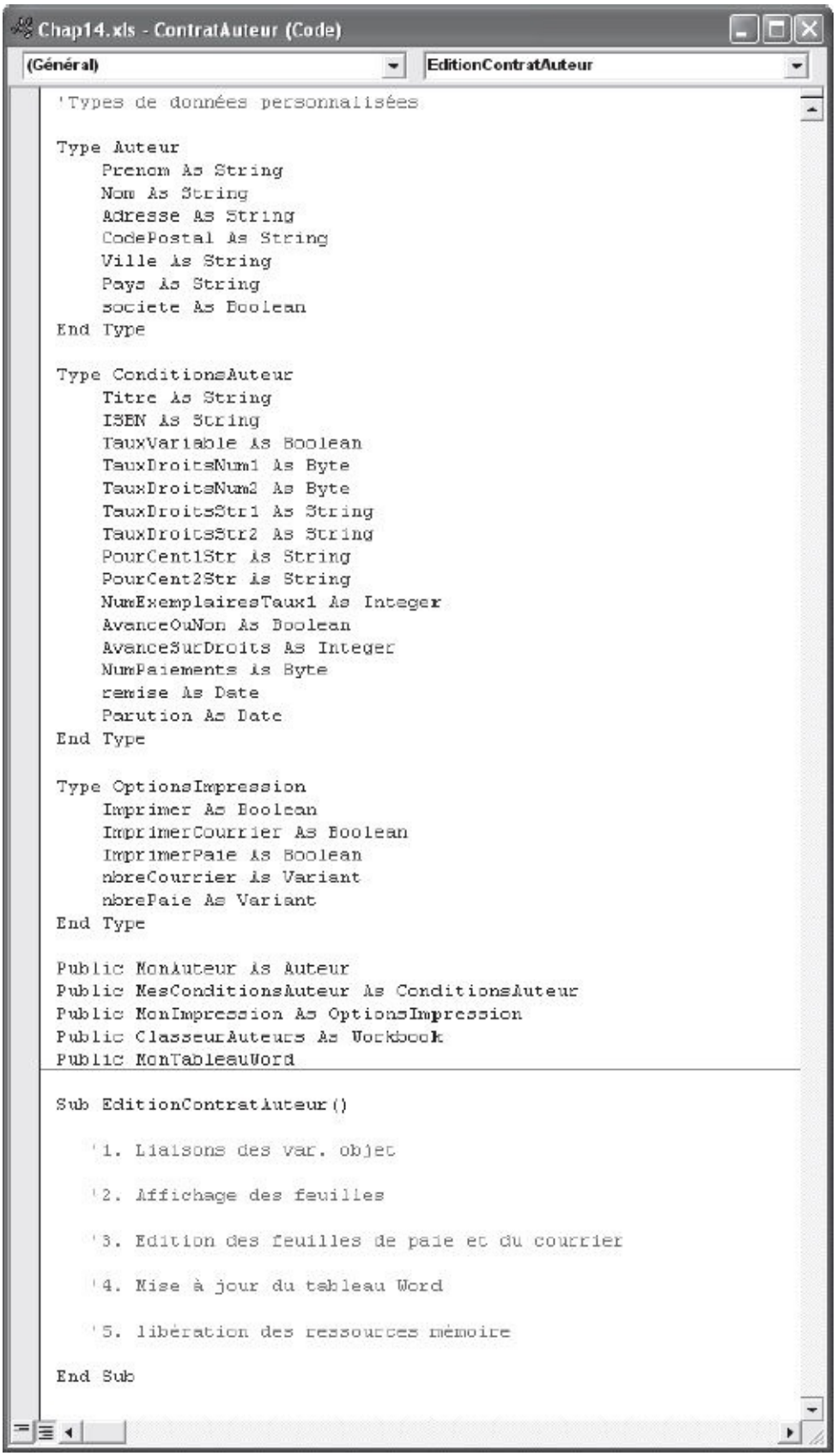

Chap14.xls - ContratAuteur (Code)
(Général)
EditionContratAuteur
'Types de données personnalisées
Type Auteur
Prenom As String
Nom As String
Adresse As String
CodePostal As String
Ville As String
Pays As String
societe As Boolean
End Type
Type ConditionsAuteur
Titre As String
ISBN As String
TauxVariable As Boolean
TauxDroitsNum1 As Byte
TauxDroitsNum2 As Byte
TauxDroitsStr1 As String
TauxDroitsStr2 As String
PourCent1Str As String
PourCent2Str As String
NumExemplairesTaux1 As Integer
AvanceOuNon As Boolean
AvanceSurDroits As Integer
NumPaiements As Byte
remise As Date
Parution As Date
End Type
Type OptionsImpression
Imprimer As Boolean
ImprimerCourrier As Boolean
ImprimerPaie As Boolean
nbreCourrier As Variant
nbrePaie As Variant
End Type
Public MonAuteur As Auteur
Public MesConditionsAuteur As ConditionsAuteur
Public MonImpression As OptionsImpression
Public ClasseurAuteurs As Workbook
Public MonTableauWord
Sub EditionContratAuteur()
'1. Liaisons des var. objet
'2. Affichage des feuilles
'3. Edition des feuilles de paie et du courrier
'4. Mise à jour du tableau Word
'5. libération des ressources mémoire
End Sub

Figure 17-5 – *Le module ContratAuteur contient les déclarations de types et le corps de la procédure principale.*

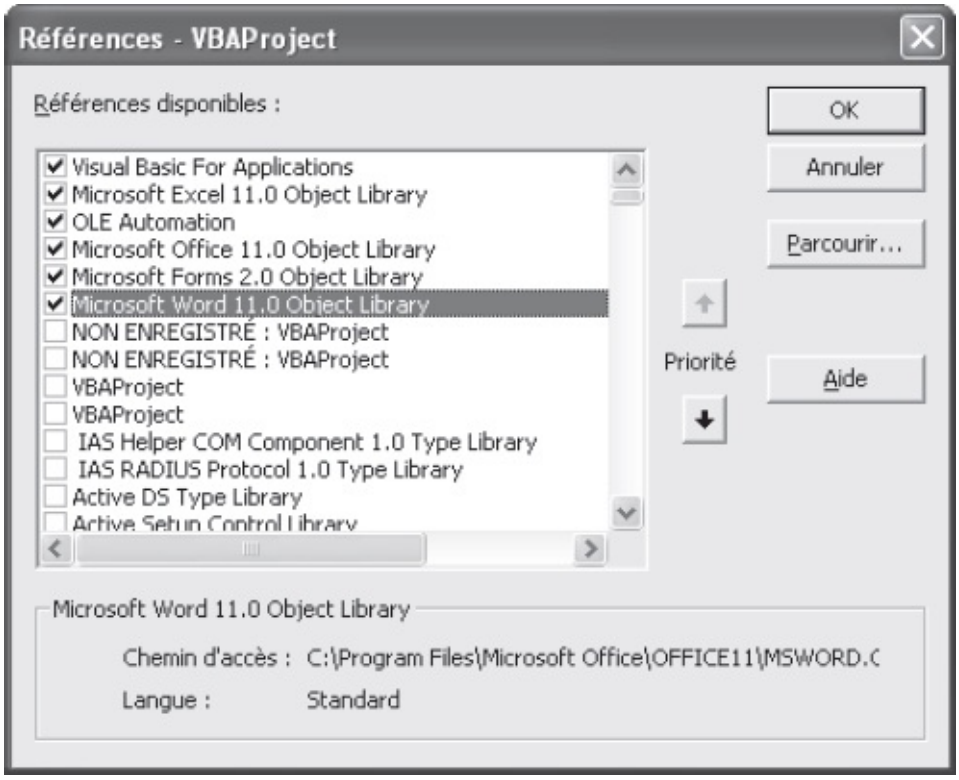


Figure 17-6 – *Créez une référence vers la bibliothèque d'objets de Word.*

Ajoutez les liaisons des variables objets dans la procédure `EditionContratAuteur`.

```
Sub EditionContratAuteur()
   '1. Liaisons des variables objets
   Set ClasseurAuteurs = GetObject("C:\Mes documents\Classeur Auteurs.xlsx")
   Set MonTableauWord = GetObject(, "Word.Application")
   If Err.Number<>0 Then Err.Clear


   '2. Affichage des feuilles
   '3. Edition des feuilles de paie et du courrier
   '4. Mise à jour du tableau Word
   '5. Libération des ressources mémoire
End Sub
```

Attention

Veillez à personnaliser les chemins d'accès aux fichiers utilisés dans cet exemple.

Créer un modèle Excel

Nous allons maintenant créer le modèle Excel qu'utilisera le programme pour

générer les feuilles de paie.

Créez un nouveau classeur Excel contenant une seule feuille, nommée « Auteur ». Entrez ensuite les données fixes, puis les formules comme indiqué à la figure 17-7. Formatez les cellules (bordures, polices, formats, etc.). Définissez un format numérique à deux décimales et un format de date selon le type d'information que doivent recevoir les cellules. Enregistrez ensuite le classeur en tant que modèle (extension .xlt), sous le nom `FeuillePaieAuteur`, puis fermez-le.

Conseil

Testez votre feuille de calcul. Entrez une valeur dans la cellule F15 et voyez si les cellules liées sont mises à jour avec les valeurs appropriées.

Nous n'entrerons pas ici dans le détail des formules et la façon de calculer les droits d'auteur. Notez simplement que la CSG et le RDS sont calculés sur une base de 95 % du montant brut et que le montant net se calcule de la façon suivante :

Net = Brut + TVA – AGESSA – CSG – RDS

La « Contribution à verser » et la TVA (lignes 26 et 27) sont des taxes dues par l'éditeur.

Le tableau 17-1 présente les informations qui devront être complétées par le programme dans la feuille de paie.

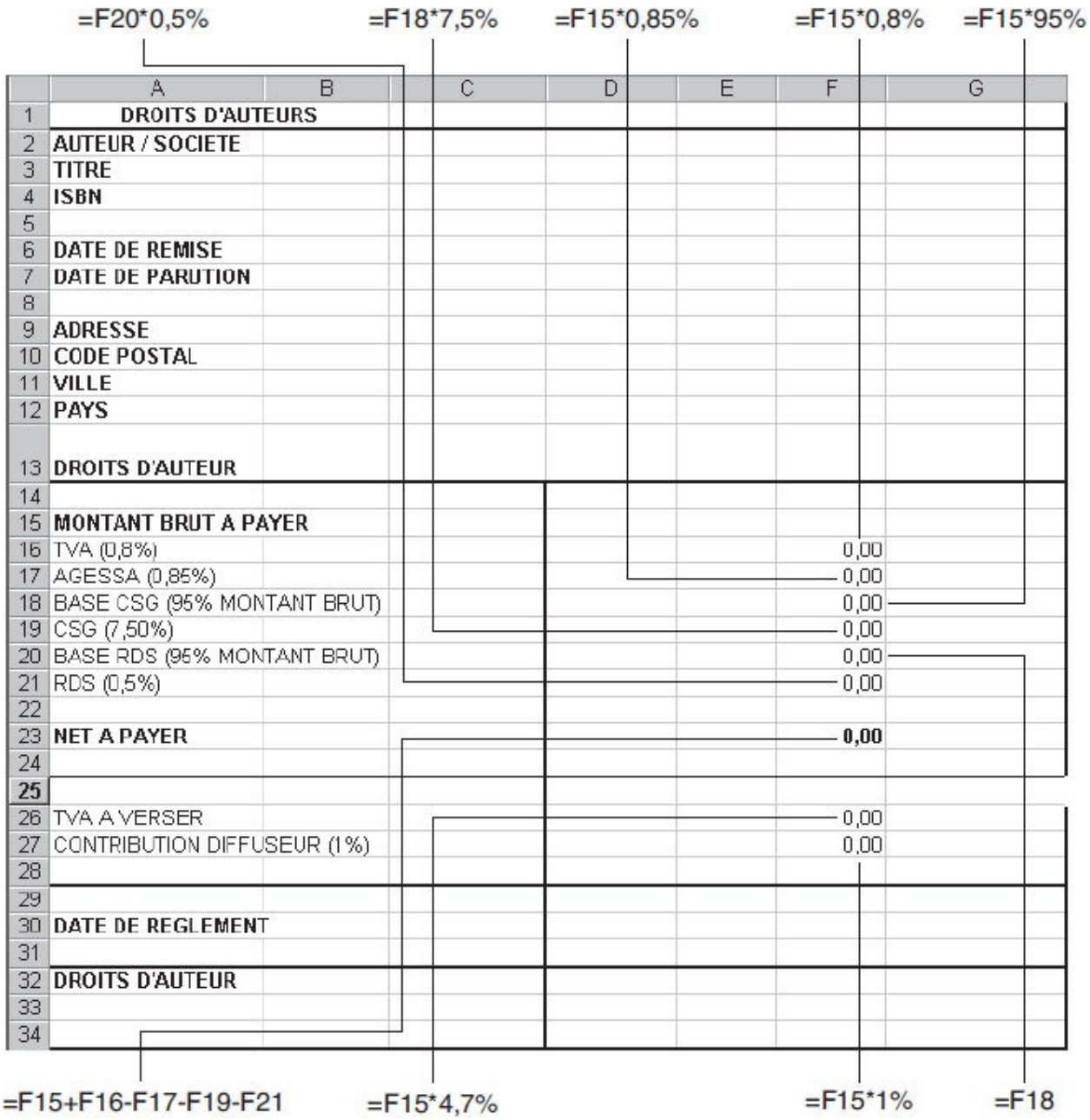


Figure 17-7 – *Le modèle FeuillePaieAuteur complété.*

Tableau 17-1. Définition des cellules qui devront être renseignées par le programme

Cellule	Contenu
C2	Auteur (Prénom + Nom) ou Société
C3	Titre de l'ouvrage
C4	ISBN
C6	Date de remise

C7	Date de parution
C9	Adresse
C10	Code postal
C11	Ville
C12	Pays
F15	Montant de l'avance sur droits à payer
F30	Date de règlement
B32	Nombre d'exemplaires touchés par le taux 1
F32	Taux 1 sous forme de valeur numérique
F33	Taux 2 sous forme de valeur numérique
G32	Taux 1 sous forme de chaîne
G33	Taux 2 sous forme de chaîne

Définir et créer des interfaces

Le programme proposera 5 interfaces qui rassembleront les différentes informations requises pour éditer le contrat :

- `fmContratAuteur`. Choix d'un auteur dans la liste extraite du fichier Excel.
- `fmContratConditions`. Fourniture des informations concernant l'ouvrage (titre et ISBN) et les conditions du contrat (taux des droits d'auteur et avance).
- `fmContratDates`. Dates de remise et de parution.
- `fmContratImpression`. Définition des documents à imprimer et du nombre d'exemplaires.
- `fmContratFin`. Message indiquant à l'utilisateur que les informations nécessaires à l'édition du contrat ont été recueillies.

Toutes les feuilles, à l'exception de la première, proposeront un bouton de commande permettant de réafficher la feuille précédente. Toutes les feuilles contiendront un bouton de validation et un bouton d'Annulation.

Feuille fmContratAuteur

La feuille `fmContratAuteur` contiendra une liste modifiable pour sélectionner le nom de l'auteur parmi ceux du classeur Excel Répertoire Auteurs.xlsx. Elle contiendra également cinq zones de texte qui indiqueront respectivement le prénom, l'adresse, le nom, le code postal, la ville et le pays. Une case à cocher précisera s'il s'agit d'une société. Outre les boutons Suite et Annuler, un

bouton permettra d'ajouter les coordonnées d'une personne au fichier Excel des auteurs. La figure 17-8 présente la feuille `fmContratAuteur` en mode Conception.

> **Info**
> Afin de mettre en valeur les différentes phases d'écriture des procédures événementielles d'une feuille, nous détaillerons les procédures de la feuille `fmContratAuteur` pour chaque contrôle. Pour les feuilles suivantes, nous présenterons l'ensemble des procédures en un seul listing que nous commenterons.

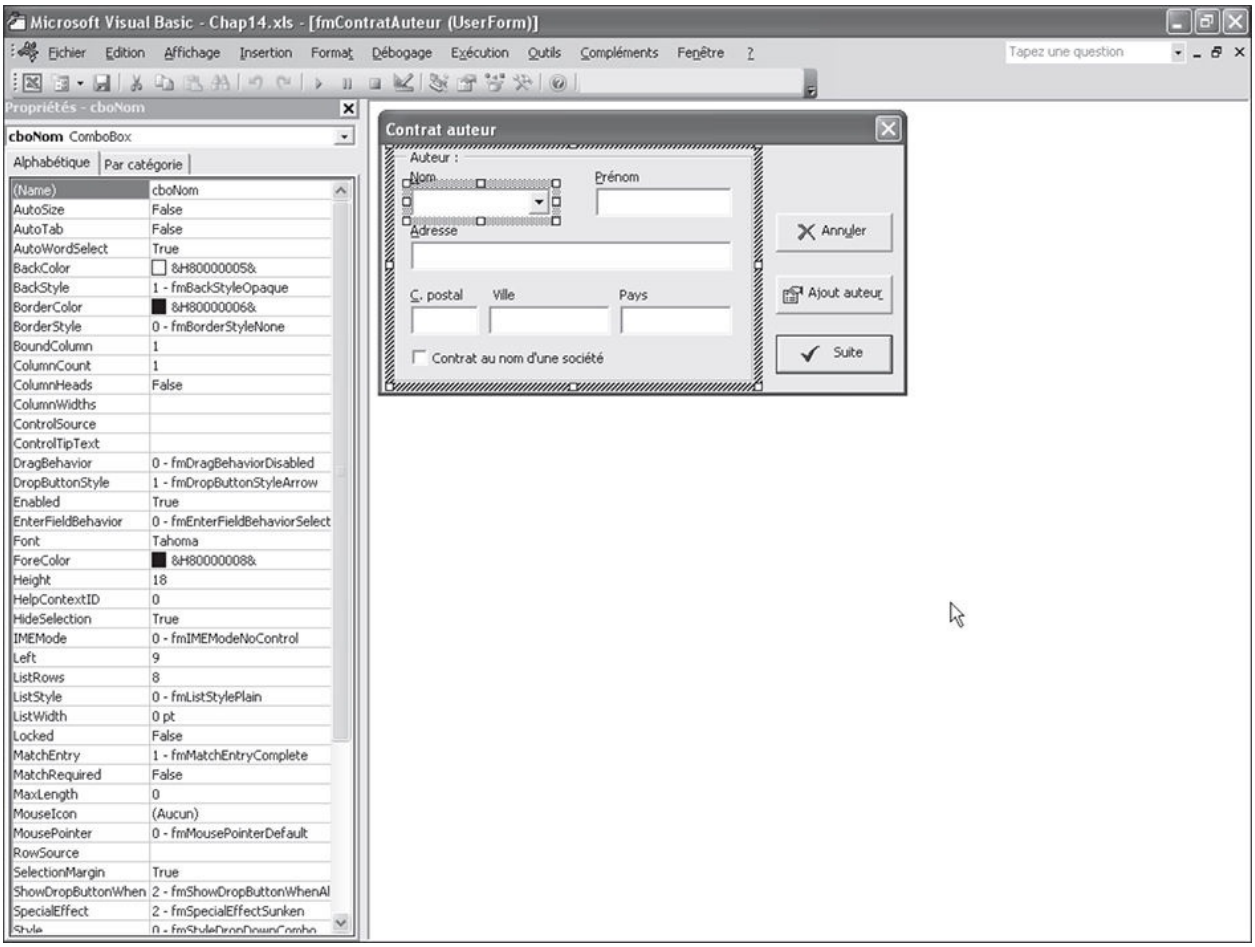


Figure 17-8 – *La feuille fmContratAuteur en mode Conception.*

Créez la feuille `fmContratAuteur` en vous fondant sur la figure 17-8. Placez un `Frame` contenant un `ComboBox`, cinq `TextBox` et un `CheckBox`. Placez un `Label` au-dessus de chaque `TextBox` et au-dessus du `ComboBox` afin d'en libeller la fonction (Nom, Prénom, Adresse, Code postal, Ville, Pays). Placez trois `CommandButton` hors du `Frame`. Affectez les propriétés suivantes aux contrôles :

Propriété	Valeur

Feuille	
Name	fmContratAuteur
Caption	Contrat auteur
Contrôle *Frame*	
Caption	Auteur :
Contrôle *ComboBox*	
Name	cboNom
Style	0 - fmStyleDropDownCombo
MatchEntry	1 - fmMatchEntryComplete
Contrôle *TextBox* 1	
Name	txtPrenom
Contrôle *TextBox* 2	
Name	txtAdresse
Contrôle *TextBox* 3	
Name	txtCodePostal
Contrôle *TextBox* 4	
Name	txtVille
Contrôle *TextBox* 5	
Name	txtPays
Contrôle *Label* 1	
Name	lbNom
Caption	Nom
Contrôle *Label* 2 [1]	
Name	lbPrenom
Caption	Prénom
Contrôle *CheckBox*	
Name	chkSociete
Caption	Contrat au nom d’une société
Contrôle *CommandButton* 1	
Name	cmdAnnuler
Caption	Annuler
Cancel	True
Contrôle *CommandButton* 2	
Name	cmdAjouterAuteur
Caption	Ajouter auteur
Contrôle *CommandButton* 3	
Name	cmdSuite
Caption	Suite
Default	True

1. Il n'est pas nécessaire de définir les propriétés `Name` des autres contrôles Label, car ils ne seront pas manipulés.

Ouvrez ensuite la fenêtre Code de la feuille et placez-y les procédures suivantes :

1. Code d'initialisation de la feuille

```
 1: Private Sub UserForm_Initialize()
 2:   Application.StatusBar = "Chargement des auteurs en cours. Veuillez
      ➥ patienter…"
 3:   Application.Cursor = xlWait
 4:   Call MiseAjourListeDeroulante
 5:   Application.StatusBar = ""
 6:   Application.Cursor = xlDefault
 7:   cboNom.SetFocus
 8: End Sub
 9:
10: Private Sub MiseAjourListeDeroulante()
11:   'Suppression des entrées de la liste si celle-ci en contient
12:   If cboNom.ListCount>=1 Then
13:     Dim ElementListe As Integer
14:     Dim NbreElt As Integer
15:     NbreElt = cboNom.ListCount - 1
16:     For ElementListe = NbreElt To 0 Step -1
17:       cboNom.RemoveItem (ElementListe)
18:     Next ElementListe
19:   End If
20:   'Ajout de tous les noms du répertoire des auteurs
21:   Dim compteur As Long
22:   Dim AjoutAuteur As String
23:   For compteur = 2 To ClasseurAuteurs.Sheets(1).Range("A1").End(xlDown).Row
24:     AjoutAuteur = ClasseurAuteurs.Sheets(1).Range("A" & compteur).Value
25:     cboNom.AddItem (AjoutAuteur)
26:   Next compteur
27: End Sub
```

La procédure d'initialisation commence par afficher un message dans la barre d'état de l'application afin d'informer l'utilisateur du chargement en cours. Ligne 3, le curseur est transformé en sablier. Ligne 4, `MiseAjourListeDeroulante` est appelée, puis la procédure appelante reprend la main, le message de la barre d'état est effacé tandis que le curseur reprend sa forme normale. Enfin, ligne 7, la liste modifiable reçoit le focus.

Attention

La variable `ClasseurAuteurs` doit avoir été déclarée et un classeur doit lui être affecté avant que la feuille ne soit affichée. Si tel n'est pas le cas, l'instruction de la ligne 23 provoquera une erreur.

La procédure `MiseAjourListeDeroulante` a pour fonction d'ajouter les noms des auteurs à la liste modifiable. Si celle-ci n'est pas vide, les instructions des lignes 11 à 19 suppriment d'abord les éléments qu'elle contient. Nous y

reviendrons plus tard. Lignes 20 à 26, les auteurs sont ajoutés à la liste. On utilise pour cela une structure `For…Next` qui ajoute un à un le contenu des cellules de la colonne A à la liste déroulante. Le compteur va de 2 (la cellule A2 étant la première à contenir un nom d'auteur) au numéro de ligne de la dernière cellule non vide dans la colonne A : `Range("A1").End(xlDown).Row`.

Conseil

À ce stade, vérifiez que la mise à jour de la liste modifiable s'effectue correctement. Exécutez pour cela la procédure `EditionContratAuteur`, après avoir pris soin d'y ajouter l'instruction d'affichage de la feuille. Celle-ci doit se présenter comme suit :

```
Sub EditionContratAuteur()
  Set ClasseurAuteurs = GetObject("C:\Mes documents\Classeur Auteurs.xlsx")
  Set MonTableauWord = GetObject(, "Word.Application")
  If Err.Number<>0 Then Err.Clear
  fmContratAuteur.Show
End Sub
```

2. Code de mise à jour des zones de texte

```
1: Private Sub cboNom_Change()
2:   Dim LigneSel As Long
3:   LigneSel = cboNom.ListIndex + 2
4:   txtPrenom = ClasseurAuteurs.Sheets(1).Range("B" & LigneSel).Value
5:   txtAdresse = ClasseurAuteurs.Sheets(1).Range("C" & LigneSel).Value
6:   txtVille = ClasseurAuteurs.Sheets(1).Range("D" & LigneSel).Value
7:   txtCodePostal = ClasseurAuteurs.Sheets(1).Range("E" & LigneSel).Value
8:   txtPays = ClasseurAuteurs.Sheets(1).Range("F" & LigneSel).Value
9: End Sub
```

Cette procédure événementielle met automatiquement à jour le contenu des zones de texte lorsque l'utilisateur change le contenu de la liste modifiable Nom (par la sélection d'un nom dans la liste ou par la saisie d'un nouveau nom). La variable `LigneSel` stocke le numéro de ligne du classeur des auteurs correspondant au nom sélectionné. La propriété `ListIndex` du contrôle `cboNom` renvoie l'index de l'élément sélectionné. On y ajoute 2 afin d'obtenir le numéro de la ligne correspondante dans le classeur Excel – le premier nom de la liste correspond à la ligne 2 et la première valeur d'index d'un `ComboBox` est 0.

Lignes 4 à 8, la valeur de chacun des `TextBox` est mise à jour avec les informations de la ligne contenant le nom. Par exemple, si l'utilisateur sélectionne un nom provenant de la cellule A10, `txtPrenom` reçoit le contenu de la cellule B10, `txtAdresse` reçoit le contenu de C10, etc.

Notez que, si l'utilisateur saisit une valeur dans le `ComboBox` (ce qui est possible, car la propriété `Style` du contrôle a été définie à `0 - fmStyleDropDownCombo`), le complément automatique apparaît si un nom correspondant aux premières

lettres saisies existe dans la liste (`MatchEntry` a été définie à `1 - fmMatchEntryComplete`) et que les zones de texte sont automatiquement mises à jour avec les valeurs correspondant à ce nom. Si l'utilisateur saisit un nom qui ne fait pas partie de la liste, la propriété `ListIndex` du contrôle renvoie la valeur – 1. `LigneSel` reçoit alors la valeur 1 (– 1 + 2) et les zones de texte affichent donc le contenu des cellules de la ligne 1, soit le titre des colonnes (voir figure 17-9).

3. Code de la case à cocher Société

```
 1: Private Sub chkSociete_Click()
 2:   If chkSociete.Value=True Then
 3:     lbNom.Caption = "Société"
 4:     lbPrenom.Visible = False
 5:     txtPrenom.Visible = False
 6:   Else
 7:     lbNom.Caption = "Nom"
 8:     lbPrenom.Visible = True
 9:     txtPrenom.Visible = True
10:   End If
11: End Sub
```

La case à cocher Société est cochée si le contrat est établi au nom d'une personne morale. Quand l'événement `Click` du contrôle est détecté, si la case est cochée (ligne 2), le `Label lbNom` voit sa propriété `Caption` redéfinie à "Société" et les contrôles `lbPrenom` et `txtPreNom` sont masqués (`Visible = False`). Si la case est décochée, la propriété `Caption` de `lbNom` est redéfinie à "Nom" et les contrôles `lbPrenom` et `txtPrenom` sont affichés (voir figure 17-10).

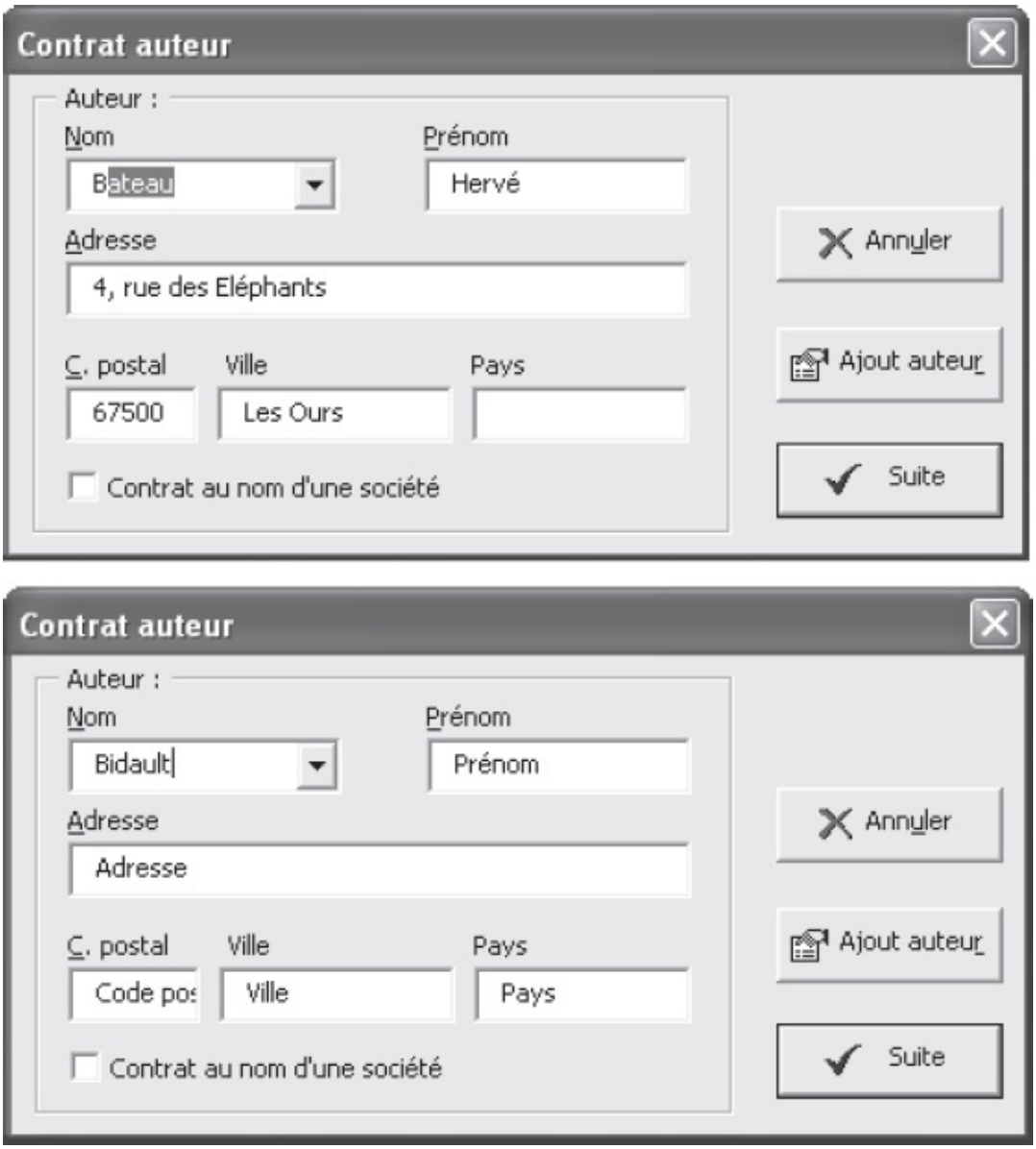


Figure 17-9 – *Lorsque l'utilisateur sélectionne un nom dans la liste ou saisit un nouveau nom, le complément automatique est proposé et les zones de texte sont mises à jour.*

Figure 17-10 – *Les libellés Nom et Prénom, ainsi que la zone de texte Prénom sont affectés par l'état de la case à cocher Société.*

4. Code d'annulation

```
 1: Private Sub cmdAnnuler_Click()
 2:   Dim rep As Byte
 3:   rep = MsgBox("Etes-vous sûr de vouloir annuler l'édition du contrat
      ➥ en cours ?", _
 4:     vbYesNo + vbQuestion, "Annuler l'édition de contrat ?")
 5:
 6:   If rep=vbYes Then
 7:     Me.Hide
 8:     Call ContratFin
 9:     End
10:   End If
```

```
11: End Sub
12:
13: Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
14:   Dim rep As Byte
15:   rep = MsgBox("Etes-vous sûr de vouloir annuler l'édition du contrat
      ➥ en cours ?", _
16:     vbYesNo + vbQuestion, "Annuler l'édition de contrat ?")
17:
18:   If rep=vbYes Then
19:     Me.Hide
20:     Call ContratFin
21:     End
22:   End If
23: End Sub
24:
25: Public Sub ContratFin()
26:   'Libérer les ressources mémoire occupées par la variable objet
27:   Set ClasseurAuteurs = Nothing
28:   'Curseur classique
29:   Application.Cursor = xlDefault
30: End Sub
```

La procédure `cmdAnnuler_Click` gère l'annulation *via* le bouton Annuler – déclenchée par un clic sur le bouton ou la frappe de la touche Échap – et `UserForm_QueryClose` gère un clic sur la case de fermeture de la fenêtre. Elles sont toutes deux semblables : la boîte de dialogue représentée à la figure 17-11 s'affiche. Si l'utilisateur clique sur le bouton Oui, la procédure `ContratFin` est appelée et l'instruction `End` met fin au programme.

Placez `ContratFin` dans le module `ContratsDivers`. Elle est déclarée publique pour que n'importe quel module du projet puisse l'appeler. Elle a pour fonction de libérer la variable objet `ClasseurAuteurs` et de redéfinir le curseur.

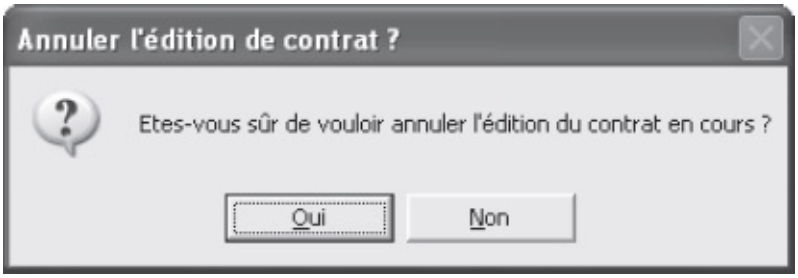


Figure 17-11 – *Demandez toujours une confirmation afin d'éviter une annulation du programme par erreur.*

5. Code du bouton Ajouter Auteur

```
1: Private Sub CmdAjouterAuteur_Click()
2:
3:   'Contrôle de la liste des auteurs
4:   Dim Ligne As Integer
5:   Application.StatusBar = "Contrôle de la liste des auteurs en cours. Veuillez
      ➥ patienter…"
6:   Application.Cursor = xlWait
```

```
 7:   For Ligne = 2 To ClasseurAuteurs.Sheets(1).Range("A1").End(xlDown).Row
 8:     If cboNom.Value=ClasseurAuteurs.Sheets(1).Range("A" & Ligne).Value And
        ➥ txtPrenom=ClasseurAuteurs.Sheets(1).Range("B" & Ligne).Value Then
 9:       Application.StatusBar = ""
10:       Application.Cursor = xlDefault
11:       Dim remplacer As Integer
12:       remplacer = MsgBox("Ce prénom et ce nom existent déjà pour un auteur.
           ➥ Remplacer par les coordonnées actuelles ?", vbOKCancel + vbCritical,
           ➥ "Remplacer les coordonnées de l'auteur ?")
13:       If remplacer=vbCancel Then
14:         Exit Sub
15:       Else
16:         Application.StatusBar = "Remplacement des coordonnées et mise à
             ➥ jour en cours. Patientez…"
17:         Application.Cursor = xlWait
18:         Call RemplacerEntreeAuteur(Ligne, cboNom.Value, txtPrenom.Value,
             ➥ txtAdresse.Value, txtVille.Value, txtCodePostal.Value, txtPays.Value)
19:         Application.StatusBar = ""
20:         Application.Cursor = xlDefault
21:         Exit Sub
22:       End If
23:     End If
24:   Next Ligne
25:   Application.StatusBar = ""
26:   Application.Cursor = xlDefault
27:
28:   'Ajout du nouvel auteur
29:   Application.StatusBar = "Ajout de l'auteur en cours. Veuillez patienter…"
30:   Application.Cursor = xlWait
31:   Call CreerEntreeAuteur(cboNom.Value, txtPrenom.Value, txtAdresse.Value,
       ➥ txtVille.Value, txtCodePostal.Value, txtPays.Value)
32:
33:   'tri et enregistrement du repertoire
34:   ClasseurAuteurs.Sheets(1).Range("A2:H"
       ➥ & ClasseurAuteurs.Sheets(1).Range("A1").End(xlDown).Row + 1).Sort
       ➥ Key1:=ClasseurAuteurs.Sheets(1).Range("A2"),
       ➥ Key2:=ClasseurAuteurs.Sheets(1).Range("B2"),
       ➥ Key3:=ClasseurAuteurs.Sheets(1).Range("C2")
35:   ClasseurAuteurs.Save
36:
37:   'Mise à jour de la liste déroulante
38:   StatusBar = "Mise à jour de la liste en cours. Veuillez patienter…"
39:   Call MiseAjourListeDeroulante
40:   Application.StatusBar = ""
41:   Application.Cursor = xlDefault
42: End Sub
43:
44:
45: Public Sub CreerEntreeAuteur(Nom As String, Prenom As String,
     ➥ Adresse As String, Ville As String, _
46:   CodePostal As String, Optional Pays As String = "")
47:   Dim LigneAjout As Integer
48:   LigneAjout = ClasseurAuteurs.Sheets(1).Range("A1").End(xlDown).Row + 1
49:
50:   ClasseurAuteurs.Sheets(1).Range("A" & LigneAjout).Value = Nom
51:   ClasseurAuteurs.Sheets(1).Range("B" & LigneAjout).Value = Prenom
52:   ClasseurAuteurs.Sheets(1).Range("C" & LigneAjout).Value = Adresse
53:   ClasseurAuteurs.Sheets(1).Range("D" & LigneAjout).Value = Ville
54:   ClasseurAuteurs.Sheets(1).Range("E" & LigneAjout).Value = CodePostal
55:   ClasseurAuteurs.Sheets(1).Range("F" & LigneAjout).Value = Pays
56: End Sub
57:
58:
59: Public Sub RemplacerEntreeAuteur(Ligne As Integer, Nom As String,
```

```
     ➥ Prenom As String, Adresse As String, _
60:   Ville As String, CodePostal As String, Optional Pays As String = "")
61:
62:   ClasseurAuteurs.Sheets(1).Range("A" & Ligne).Value = Nom
63:   ClasseurAuteurs.Sheets(1).Range("B" & Ligne).Value = Prenom
64:   ClasseurAuteurs.Sheets(1).Range("C" & Ligne).Value = Adresse
65:   ClasseurAuteurs.Sheets(1).Range("D" & Ligne).Value = Ville
66:   ClasseurAuteurs.Sheets(1).Range("E" & Ligne).Value = CodePostal
67:   ClasseurAuteurs.Sheets(1).Range("F" & Ligne).Value = Pays
68:
69:   ClasseurAuteurs.Sheets(1).Range("A2:H"
       ➥ & ClasseurAuteurs.Sheets(1).Range("A1").End(xlDown).Row).Sort _
70:     Key1:=ClasseurAuteurs.Sheets(1).Range("A2"), _
71:     Key2:=ClasseurAuteurs.Sheets(1).Range("B2"), _
72:     Key3:=ClasseurAuteurs.Sheets(1).Range("C2")
73:
74:   ClasseurAuteurs.Save
75: End Sub
```

`CmdAjouterAuteur_Click` (lignes 1 à 42) fait appel à deux procédures publiques : `CreerEntreeAuteur` (lignes 45 à 56) et `RemplacerEntreeAuteur` (lignes 59 à 75). Celles-ci seront stockées dans le module `RepertoireAuteurs` puisque leur fonction est de manipuler le fichier Excel des auteurs.

`CmdAjouterAuteur_Click` commence par vérifier s'il existe déjà une entrée dans le classeur avec le même nom et la même adresse (lignes 3 à 24). Selon le cas, l'une des deux procédures `CreerEntreeAuteur` ou `RemplacerEntreeAuteur` est appelée.

Le contrôle s'effectue à l'aide d'une boucle `For…Next` (lignes 7 à 24) qui contrôle les entrées du classeur de la ligne 2 à la dernière ligne contenant des données. Le curseur est préalablement transformé en sablier et un message s'affiche dans la barre d'état (lignes 5 et 6). À chaque passage de la boucle, le programme teste si les colonnes A et B de la ligne testée ont les mêmes valeurs que les contrôles `cboNom` et `txtPrenom` de l'interface (ligne 8). Si tel est le cas, la barre d'état et le curseur retrouvent leurs valeurs par défaut et l'utilisateur se voit proposer le remplacement des coordonnées de l'auteur par celles entrées dans la feuille (ligne 12), comme présenté à la figure 17-12. Si l'utilisateur choisit le bouton Annuler, l'instruction `Exit` de la ligne 14 entraîne la sortie de la procédure. S'il répond Oui, un message est à nouveau affiché dans la barre d'état, le curseur prend la forme d'un sablier (lignes 16 et 17) et `RemplacerEntreeAuteur` est appelée. Lorsque la procédure appelante reprend la main, la barre d'état et le curseur reprennent leurs valeurs par défaut, puis une instruction `Exit` entraîne la sortie de la procédure.

Figure 17-12 – *Le programme vérifie qu'il n'existe pas déjà une entrée pour l'auteur.*

Si aucune entrée semblable n'a été trouvée, le programme atteint la ligne 25 sans que rien ne se soit passé. Le nouvel auteur est alors ajouté sans qu'une intervention de l'utilisateur ne soit requise : `CreerEntreeAuteur` est appelée ligne 31, puis la procédure appelante reprend la main et le classeur est trié et sauvegardé (lignes 34 et 35). La liste modifiable est ensuite mise à jour (ligne 39). Notez que la procédure `MiseAjourListeDeroulante` commence par supprimer les entrées de la liste pour les ajouter à nouveau. Enfin, lignes 40 et 41, la barre d'état et le curseur retrouvent leur aspect par défaut.

`CreerEntreeAuteur` et `RemplacerEntreeAuteur` reçoivent toutes les deux les nom, prénom, adresse, code postal, ville et pays entrés par l'utilisateur dans l'interface. `RemplacerEntreeAuteur` reçoit en plus le numéro de la ligne à remplacer (notez les appels, lignes 18 et 31, et les déclarations, lignes 45 et 59). Tandis que `CreerEntreeAuteur` ajoute les informations sur une ligne vide, `RemplacerEntreeAuteur` remplace les données de la ligne dont le numéro lui a été passé par celles fournies par l'utilisateur.

6. Code du bouton Suite

```
 1: Private Sub CmdSuite_Click()
 2:
 3:   'Vérifier que les champs ont été correctement informés
 4:   If cboNom.Value="" Then
 5:     MsgBox "Vous devez indiquer un nom d'auteur.", _
 6:       vbOKOnly + vbInformation, "Informations incomplètes"
 7:     cboNom.SetFocus
 8:     Exit Sub
 9:   ElseIf txtPrenom.Value="" And ChkSociete.Value=False Then
10:     MsgBox "Vous devez indiquer un prénom pour l'auteur." & Chr(13) _
11:       & "S'il s'agit d'un contrat au nom d'une société, cochez la case
          correspondante", _
12:       vbOKOnly + vbInformation, "Informations incomplètes"
13:     txtPrenom.SetFocus
14:     Exit Sub
15:   ElseIf txtAdresse.Value="" Then
16:     MsgBox "Vous devez indiquer une adresse.", _
17:       vbOKOnly + vbInformation, "Informations incomplètes"
18:     txtAdresse.SetFocus
19:     Exit Sub
20:   ElseIf txtCodePostal.Value="" Then
21:     MsgBox "Vous devez indiquer un code postal.", _
22:       vbOKOnly + vbInformation, "Informations incomplètes"
23:     txtCodePostal.SetFocus
24:     Exit Sub
25:   ElseIf txtVille.Value="" Then
26:     MsgBox "Vous devez indiquer une ville.", _
27:       vbOKOnly + vbInformation, "Informations incomplètes"
28:     TxtVille.SetFocus
29:     Exit Sub
30:   End If
31:
32:   'stockage des données dans des variables
33:   With MonAuteur
34:     .Nom = cboNom.Value
35:     .Prenom = txtPrenom.Value
36:     .Adresse = txtAdresse.Value
37:     .CodePostal = txtCodePostal.Value
38:     .Ville = txtVille.Value
39:     If txtPays.Value<>"" Then
40:       .Pays = Chr(13) + txtPays.Value
41:     End If
42:     .societe = chkSociete.Value
43:   End With
44:   Me.Hide
45: End Sub
```

La procédure `CmdSuite_Click` vérifie la validité des informations fournies par l'utilisateur et affecte ces valeurs aux variables appropriées. Lignes 3 à 30, une instruction conditionnelle est utilisée pour vérifier que chaque zone de texte a été complétée. Si tel n'est pas le cas, un message s'affiche à l'attention de l'utilisateur (voir figure 17-13), le focus est passé au contrôle non renseigné et une instruction `Exit` entraîne la sortie de la procédure. Notez l'expression conditionnelle de la ligne 9 qui vérifie si le prénom n'est pas fourni ET si la case Société n'est pas cochée.

Informations incomplètes
Vous devez indiquer une adresse.
OK
Informations incomplètes
Vous devez indiquer un prénom pour l'auteur.
S'il s'agit d'un contrat au nom d'une société, cochez la case correspondante
OK

Figure 17-13 – *Les données sont vérifiées avant d'être affectées aux variables.*

Si les données fournies sont valides, la structure `With…End With` des lignes 33 à 43 affecte les valeurs fournies aux espaces de stockage appropriés de la variable publique `MonAuteur`. Notez que, si le champ Pays n'est pas renseigné (s'il s'agit de la France), l'espace `MonAuteur.Pays` ne reçoit pas de valeur ; dans le cas contraire, il reçoit la valeur fournie précédée d'un retour chariot. Vous verrez plus tard pourquoi. Ligne 44, la feuille est masquée.

Feuille fmContratConditions

La feuille `fmContratConditions` contiendra, un contrôle `Frame` libellé Ouvrage et un autre libellé Conditions de rémunération. Le premier contiendra deux `Label` identifiant deux zones de texte destinées à entrer le nom de l'ouvrage et son ISBN. Le deuxième contiendra deux autres cadres, l'un libellé Droits d'auteur et l'autre libellé Avance. Le cadre Droits d'auteur contiendra les contrôles précisant si le taux est variable (`CheckBox`), le taux initial (`TextBox`, `Label` et `SpinButton`), l'éventuel deuxième taux (`TextBox`, `Label` et `SpinButton`) et au-delà de combien d'exemplaires celui-ci s'applique (`TextBox` et deux `Label`). Le cadre Avance indiquera s'il y a ou non une avance sur droits d'auteur (`CheckBox`) et, dans l'affirmative, la valeur de cette avance (`TextBox` et deux `Label`) et le nombre de versements (deux `OptionButton`). Outre les boutons Annuler et Suite, un bouton Retour permettra de revenir à la feuille précédente.

Créez la feuille `fmContratConditions` en vous fondant sur les figures 17-14 et 17-15, ainsi que sur le tableau suivant.

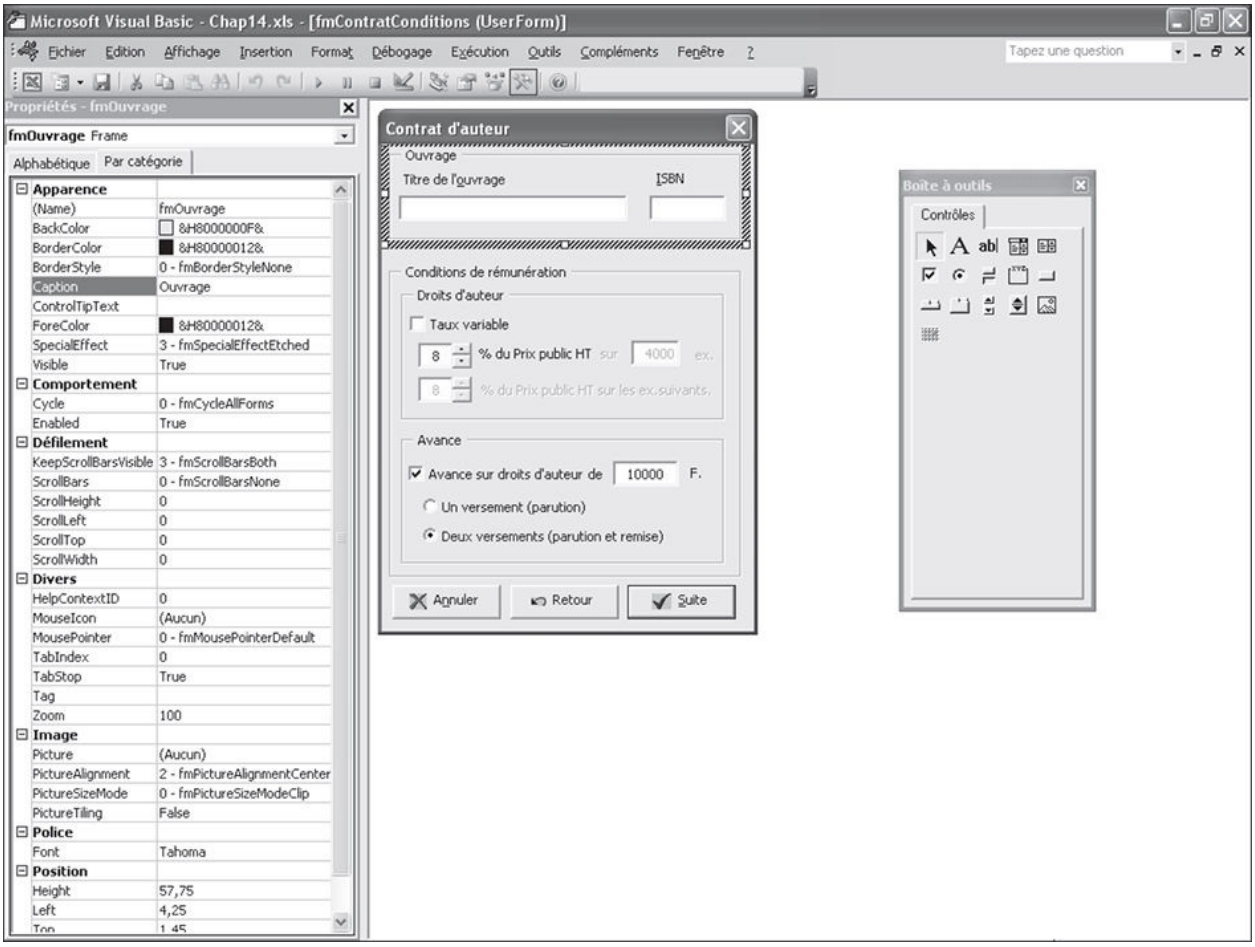


Figure 17-14 – *La feuille fmContratConditions en mode Conception.*

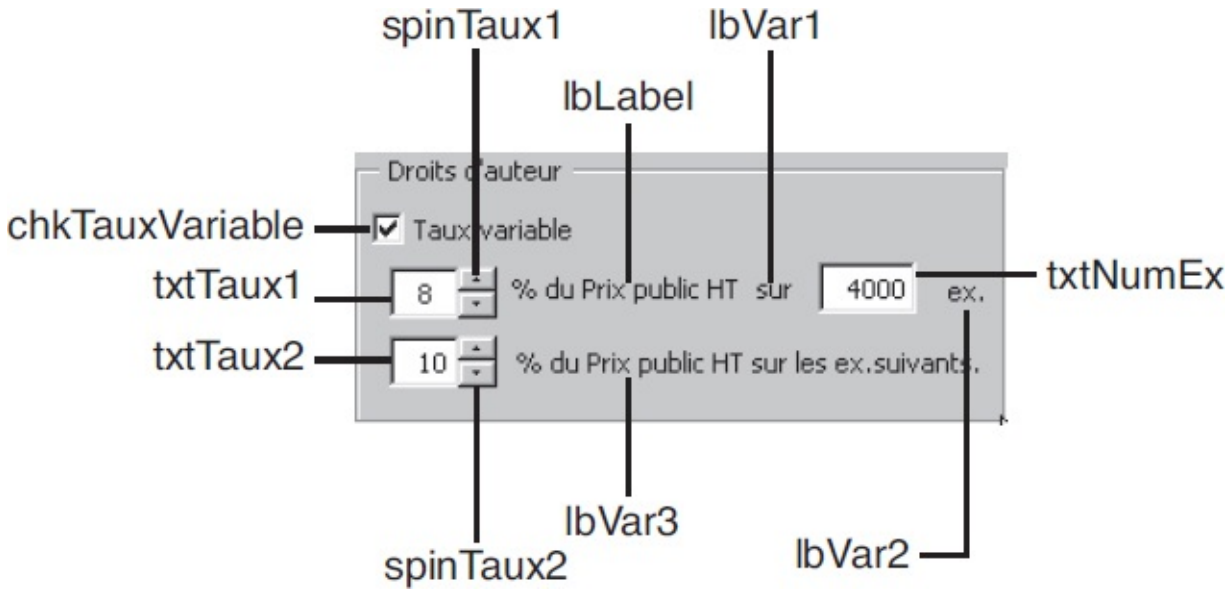


Figure 17-15 – *Le cadre Droits d'auteur de la feuille fmContratAuteur.*

Propriété	Valeur
Feuille	
Name	fmContratConditions
Caption	Contrat d’auteur
Contrôle *Frame* 1	
Caption	Ouvrage
Contrôle *Frame* 2	
Caption	Conditions de rémunération
Contrôle *Frame* 2.1	
Caption	Droits d’auteur
Contrôle *Frame* 2.2	
Caption	Avance
Contrôles du* Frame *1	
Contrôle *TextBox* 1	
Name	txtTitre
Contrôle *TextBox* 2	
Name	txtISBN
Contrôle *Label* 1	
Caption	Titre de l’ouvrage
Contrôle *Label* 2	
Caption	ISBN
Contrôles du* Frame *2.1	
Contrôle *CheckBox*	
Name	chkTauxVariable
Caption	Taux variable
Value	False
Contrôle *TextBox* 1	
Name	txtTaux1
Value	8
Contrôle *TextBox* 2	
Name	txtTaux2
Value	8
Contrôle *TextBox* 3	
Name	txtNumEx
Value	4000
Enabled	False
Contrôle *SpinButton* 1	
Name	spinTaux1
Value	8

Min	1
Max	20
Contrôle *SpinButton* 2	
Name	spinTaux2
Value	8
Min	1
Max	20
Enabled	False
Contrôle *Label* 1	
Name	Label1
Caption	% du prix public HT
Enabled	True
Contrôle *Label* 2	
Name	lbVar1
Caption	sur
Enabled	False
Contrôle *Label* 3	
Name	lbVar2
Caption	sur
Enabled	False
Contrôle *Label* 4	
Name	lbVar3
Caption	% du prix HT sur les ex. suivants
Enabled	False
Contrôles du* Frame *2.2	
Contrôle *CheckBox*	
Name	chkAvance
Caption	Avances sur droits d'auteur
Value	True
Contrôle *TextBox* 1	
Name	txtAvance
Value	1000
Contrôle *Label* 1	
Name	lbAvance1
Caption	de
Contrôle *Label* 2	
Name	lbAvance2
Caption	euros
Contrôle *OptionButton* 1	

Name	optUnVersement
Caption	Un versement (parution)
Value	False
Contrôle *OptionButton* 2	
Name	optDeuxVersements
Caption	Deux versements (parution et remise)
Value	True
Boutons de commande	
Contrôle *CommandButton* 1	
Name	cmdAnnuler
Caption	Annuler
Cancel	True
Contrôle *CommandButton* 2	
Name	cmdRetour
Caption	Retour
Contrôle *CommandButton* 3	
Name	CmdSuite
Caption	Suite
Default	True

Copiez ensuite les instructions suivantes dans la fenêtre Code de la feuille :

```
 1: Private Sub UserForm_Initialize()
 2:   txtTitre.SetFocus
 3: End Sub
 4:
 5: Private Sub chkTauxVariable_Click()
 6:   If chkTauxVariable.Value=True Then
 7:     lbVar1.Enabled = True
 8:     lbVar2.Enabled = True
 9:     lbVar3.Enabled = True
10:     txtTaux2.Enabled = True
11:     spinTaux2.Enabled = True
12:     txtNumEx.Enabled = True
13:   Else
14:     lbVar1.Enabled = False
15:     lbVar2.Enabled = False
16:     lbVar3.Enabled = False
17:     txtTaux2.Enabled = False
18:     spinTaux2.Enabled = False
19:     txtNumEx.Enabled = False
20:   End If
21: End Sub
22:
23: Private Sub txtTaux1_Change()
24:   If IsNumeric(txtTaux1.Value)=False Then
25:     MsgBox "Le taux pour les droits d'auteur doit être une valeur
        ➥ numérique.", _
26:       vbOKOnly + vbInformation, "Informations incomplètes"
27:     txtTaux1.Value = ""
28:     txtTaux1.SetFocus
```

```
29:     Exit Sub
30:   ElseIf txtTaux1.Value<1 Or txtTaux1.Value>20 Then
31:     MsgBox "Le taux pour les droits d'auteur doit être compris entre 1
        ➥ et 20.", _
32:       vbOKOnly + vbInformation, "Valeur non valide"
33:     txtTaux1.Value = ""
34:     txtTaux1.SetFocus
35:   End If
36:   spinTaux1.Value = txtTaux1.Value
37: End Sub
38:
39: Private Sub txtTaux2_Change()
40:   If IsNumeric(txtTaux2.Value)=False Then
41:     MsgBox "Le taux pour les droits d'auteur doit être une valeur
        ➥ numérique.", _
42:       vbOKOnly + vbInformation, "Informations incomplètes"
43:     txtTaux2.Value = ""
44:     txtTaux2.SetFocus
45:     Exit Sub
46:   ElseIf txtTaux2.Value<1 Or txtTaux2.Value>20 Then
47:     MsgBox "Le taux pour les droits d'auteur doit être compris entre 1
        ➥ et 20.", _
48:       vbOKOnly + vbInformation, "Valeur non valide"
49:     txtTaux2.Value = ""
50:     txtTaux2.SetFocus
51:     Exit Sub
52:     spinTaux2.Value = txtTaux1.Value
53:    End If
54: End Sub
55:
56: Private Sub spinTaux1_Change()
57:   txtTaux1.Value = spinTaux1.Value
58: End Sub
59:
60: Private Sub spinTaux2_Change()
61:   txtTaux2.Value = spinTaux2.Value
62: End Sub
63:
64: Private Sub chkAvance_Click()
65:   If chkAvance.Value=True Then
66:     lbAvance1.Visible = True
67:     lbAvance2.Visible = True
68:     txtAvance.Visible = True
69:     optUnVersement.Enabled = True
70:     optDeuxVersements.Enabled = True
71:   Else
72:     lbAvance1.Visible = False
73:     lbAvance2.Visible = False
74:     txtAvance.Visible = False
75:     optUnVersement.Enabled = False
76:     optDeuxVersements.Enabled = False
77:   End If
78: End Sub
79:
80: Private Sub cmdAnnuler_Click()
81:   Dim rep As Byte
82:   rep = MsgBox("Etes-vous sûr de vouloir annuler l'édition du contrat en
      ➥ cours ?", _
83:     vbYesNo + vbQuestion, "Annuler l'édition de contrat ?")
84:   If rep=vbNo Then
85:     Exit Sub
86:   End If
87:   Me.Hide
88:   Call ContratFin
```

```
 89:   End
 90: End Sub
 91:
 92: Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
 93:   Dim rep As Byte
 94:   rep = MsgBox("Etes-vous sûr de vouloir annuler l'édition du contrat en
       ➥ cours ?", _
 95:     vbYesNo + vbQuestion, "Annuler l'édition de contrat ?")
 96:   If rep=vbNo Then
 97:     Exit Sub
 98:   End If
 99:   Me.Hide
100:   Call ContratFin
101:   End
102: End Sub
103:
104: Private Sub cmdRetour_Click()
105:   fmContratAuteur.Show
106: End Sub
107:
108: Private Sub cmdSuite_Click()
109:
110:   'Vérifier la validité des informations
111:   If txtTitre.Value="" Then
112:     MsgBox "Vous devez indiquer un titre d'ouvrage.", _
113:       vbOKOnly + vbInformation, "Informations incomplètes"
114:     txtTitre.SetFocus
115:     Exit Sub
116:   End If
117:
118:   If txtISBN.Value="" Then
119:     MsgBox "Vous devez indiquer un ISBN.", _
120:       vbOKOnly + vbInformation, "Informations incomplètes"
121:     txtISBN.SetFocus
122:     Exit Sub
123:   End If
124:   If Len(txtISBN.Value)<>6 Then
125:     MsgBox "L'ISBN n'est pas correctement formaté.", _
126:       vbOKOnly + vbInformation, "Informations incomplètes"
127:     txtISBN.SetFocus
128:     Exit Sub
129:   End If
130:
131:   If txtTaux1.Value="" Then
132:     MsgBox "Vous devez indiquer un taux pour les droits d'auteur.", _
133:       vbOKOnly + vbInformation, "Informations incomplètes"
134:     txtTaux1.SetFocus
135:     Exit Sub
136:   End If
137:
138:   If txtTaux2.Value="" And chkTauxVariable=True Then
139:     MsgBox "Vous devez indiquer un taux pour les droits d'auteur.", _
140:       vbOKOnly + vbInformation, "Informations incomplètes"
141:     txtTaux2.SetFocus
142:     Exit Sub
143:   End If
144:
145:   If txtNumEx.Value="" And chkTauxVariable=True Then
146:     MsgBox "Vous devez indiquer une valeur valide pour le nombre d'exemplaires
        ➥ affectés par le taux 1.", _
147:       vbOKOnly + vbInformation, "Informations incomplètes"
148:     txtNumEx.SetFocus
149:     Exit Sub
150:   End If
```

```
151:   If IsNumeric(txtNumEx.Value)=False And chkTauxVariable=True Then
152:     MsgBox "Le nombre d'exemplaires concernés par le taux doit être une valeur
         ➥ numérique.", _
153:       vbOKOnly + vbInformation, "Informations incomplètes"
154:     txtNumEx.Value = ""
155:     txtNumEx.SetFocus
156:     Exit Sub
157:   End If
158:
159:   If txtAvance.Value="" And chkAvance=True Then
160:     MsgBox "Vous devez indiquer une somme pour l'avance sur droits
         ➥ d'auteur.", _
161:       vbOKOnly + vbInformation, "Informations incomplètes"
162:     txtAvance.SetFocus
163:     Exit Sub
164:   End If
165:   If IsNumeric(txtAvance.Value)=False And chkAvance=True Then
166:     MsgBox "L'avance sur droits d'auteur doit être une valeur numérique.", _
167:       vbOKOnly + vbInformation, "Informations incomplètes"
168:     txtAvance.Value = ""
169:     txtAvance.SetFocus
170:     Exit Sub
171:   End If
172:
173:   'Validation des données
174:   With MesConditionsAuteur
175:     .Titre = txtTitre.Value
176:     .ISBN = txtISBN.Value
177:     .TauxVariable = chkTauxVariable.Value
178:     .TauxDroitsNum1 =txtTaux1.Value
179:     .TauxDroitsNum2 = txtTaux2.Value
180:     .NumExemplairesTaux1 = txtNumEx.Value
181:     .AvanceOuNon = chkAvance.Value
182:     If MesConditionsAuteur.AvanceOuNon=True Then
183:       MesConditionsAuteur.AvanceSurDroits = txtAvance.Value
184:     End If
185:     If optUnVersement=True Then
186:       MesConditionsAuteur.NumPaiements = 1
187:     Else
188:       MesConditionsAuteur.NumPaiements = 2
189:     End If
190:     'Obtention des équivalents chaînes des valeurs
191:     MesConditionsAuteur.TauxDroitsStr1 = RenvoyerChaîneValeur(.TauxDroitsNum1)
192:     MesConditionsAuteur.TauxDroitsStr2 = RenvoyerChaîneValeur(.TauxDroitsNum2)
193:   End With
194:
195:   Me.Hide
196: End Sub
```

La procédure d'initialisation (lignes 1 à 3) a pour seule fonction de passer le focus au contrôle `txtTitre`.

La procédure `chkTauxVariable_Click` (lignes 5 à 21) gère l'événement clic survenant sur la case à cocher Taux variable. Si cette dernière est cochée, les trois libellés `txtVar1`, `txtVar2` et `txtVar3` deviennent disponibles. Il en est de même pour la zone de texte `txtTaux2` et le bouton toupie qui lui est attaché (`spinTaux2`), ainsi que pour la zone de texte `txtNumEx` (lignes 7 à 12). Si la case est décochée, tous ces contrôles sont désactivés (voir figure 17-16).

Les procédures `txtTaux1_Change` et `txtTaux2_Change` (lignes 23 à 54) gèrent de façon identique les modifications apportées aux zones de texte destinées à recevoir les droits d'auteur. Lignes 24 à 29 et 40 à 45, on vérifie que la valeur entrée est un nombre. Dans le cas contraire, un message s'affiche à l'attention de l'utilisateur, la valeur du contrôle est réinitialisée et le focus lui est passé. Une instruction `Exit` entraîne la sortie de la procédure. Lignes 30 à 34 et 46 à 51, un traitement similaire est appliqué si la valeur saisie n'est pas comprise entre 1 et 20. Enfin, si la valeur entrée est valide, les instructions des lignes 36 et 52 l'affectent au bouton toupie associé.

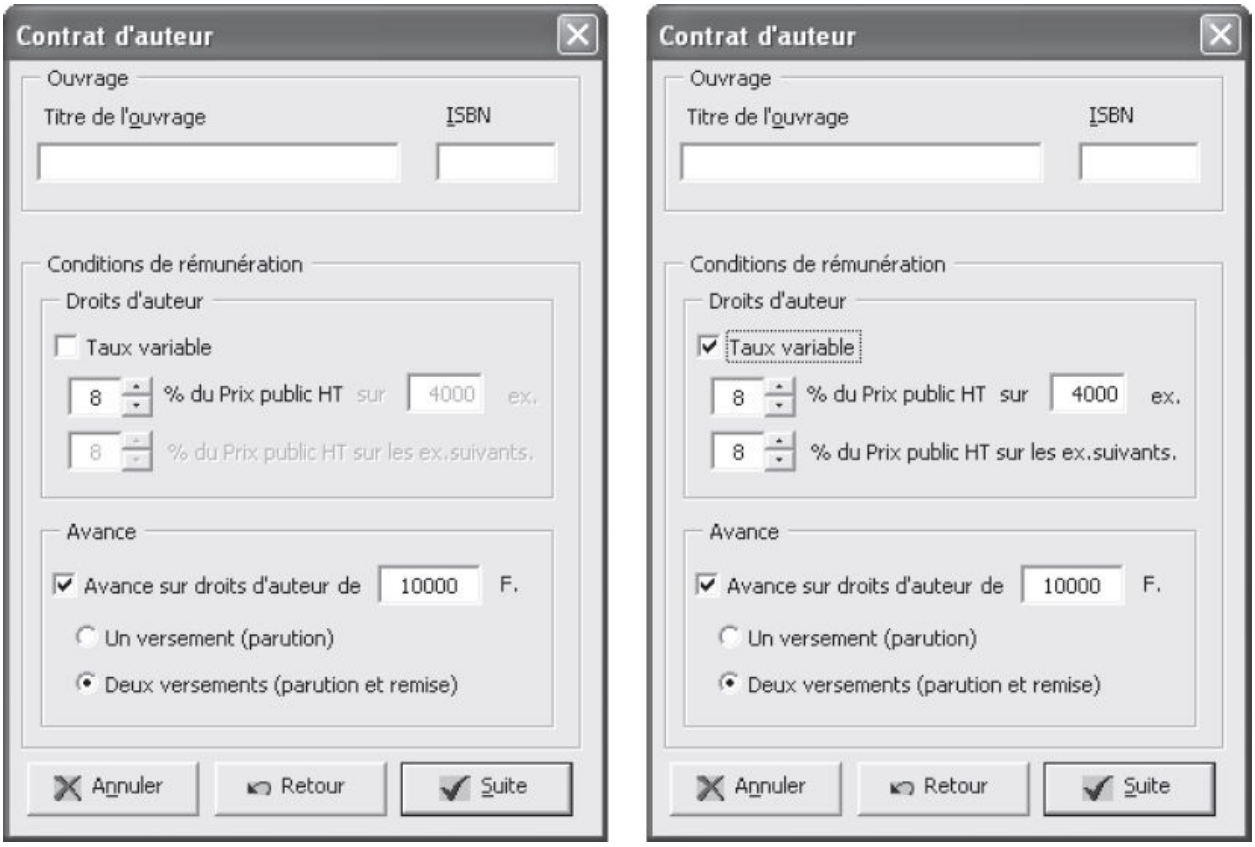


Figure 17-16 – *L'accessibilité des contrôles du cadre Droits d'auteur dépend de l'état de la case à cocher Taux variable.*

Valeur non valide
Le taux pour les droits d'auteur doit être compris entre 1 et 20.
OK
Informations incomplètes
Le taux pour les droits d'auteur doit être une valeur numérique.
OK

Figure 17-17 – *La validité des taux de droits d'auteur est contrôlée.*

Les procédures des lignes 56 à 62 affectent la valeur des boutons toupies aux zones de texte associées lorsque l'utilisateur clique sur l'un d'eux.

La procédure `chkAvance_Click` (lignes 64 à 78) gère les clics sur la case à cocher Avance sur droits d'auteur. Si cette dernière est cochée, les libellés `lbAvance1` et `lbAvance2` sont visibles (lignes 66 et 67). Il en est de même pour la zone de texte `txtAvance`. Enfin, les deux boutons d'option deviennent accessibles (lignes 69 et 70). Si la case est décochée, le traitement inverse est appliqué à ces contrôles (voir figure 17-18).

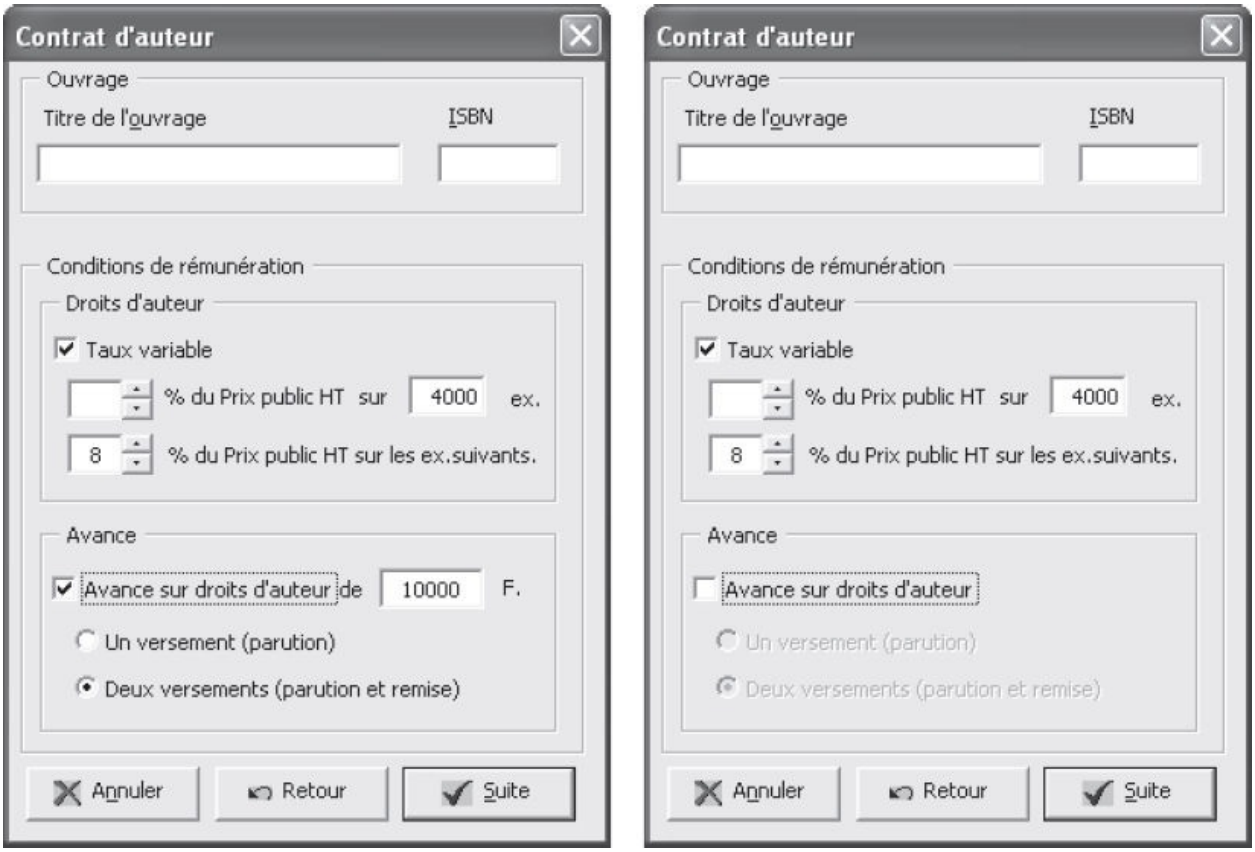


Figure 17-18 – *L'état de la case à cocher Avance sur droits d'auteur influe sur les propriétés des autres contrôles du cadre.*

Lignes 80 à 102, les procédures `CmdAnnuler_Click` et `UserForm_QueryClose` gèrent les annulations. Elles sont semblables aux procédures de la feuille `fmContratAuteur` portant le même nom.

La procédure `CmdRetour_Click` des lignes 104 à 106 gère les clics sur le bouton Retour. Elle affiche la feuille `fmContratAuteur`. Notez que la feuille `fmContratConditions` n'est pas masquée. Si vous souhaitez la masquer, ajoutez simplement l'instruction `Me.Hide` devant `fmContratAuteur.Show`.

Enfin, la procédure `CmdSuite_Click` gère les clics sur le bouton Suite. Lignes 110 à 171, la validité des données entrées est contrôlée. Le reste de la procédure les affecte à la variable `MesConditionsAuteur`.

Lignes 111 à 123, on vérifie que les zones de texte Titre et ISBN contiennent des informations. Si tel n'est pas le cas, un message s'affiche à l'attention de l'utilisateur, le focus est passé au contrôle vide et une instruction `Exit` entraîne la sortie de la procédure. Ligne 124 à 129, on vérifie que l'ISBN entré contient six caractères. Lignes 131 à 136, on vérifie qu'un taux a été précisé pour les

droits d'auteur dans le contrôle `txtTaux1`. L'instruction conditionnelle des lignes 138 à 143 vérifie que, si la case Taux variable est cochée, le second taux a bien été entré et est un nombre (lignes 151 à 157). Enfin, lignes 159 à 170, on vérifie que, si la case Avance est cochée, une valeur a été fournie (lignes 159 à 164) et est un nombre (lignes 165 à 171).

Si les données fournies sont valides, la procédure atteint la ligne 173 sans qu'une instruction `Exit` n'en ait entraîné la sortie. Les données sont alors affectées aux différents sous-types de la variable `MesConditionsAuteur` à l'aide d'une structure `With…End With`. Remarquez l'instruction conditionnelle utilisée lignes 185 à 189 pour affecter la valeur 1 ou 2 à `MesConditionsAuteur`, selon que l'un ou l'autre des boutons d'option est coché.

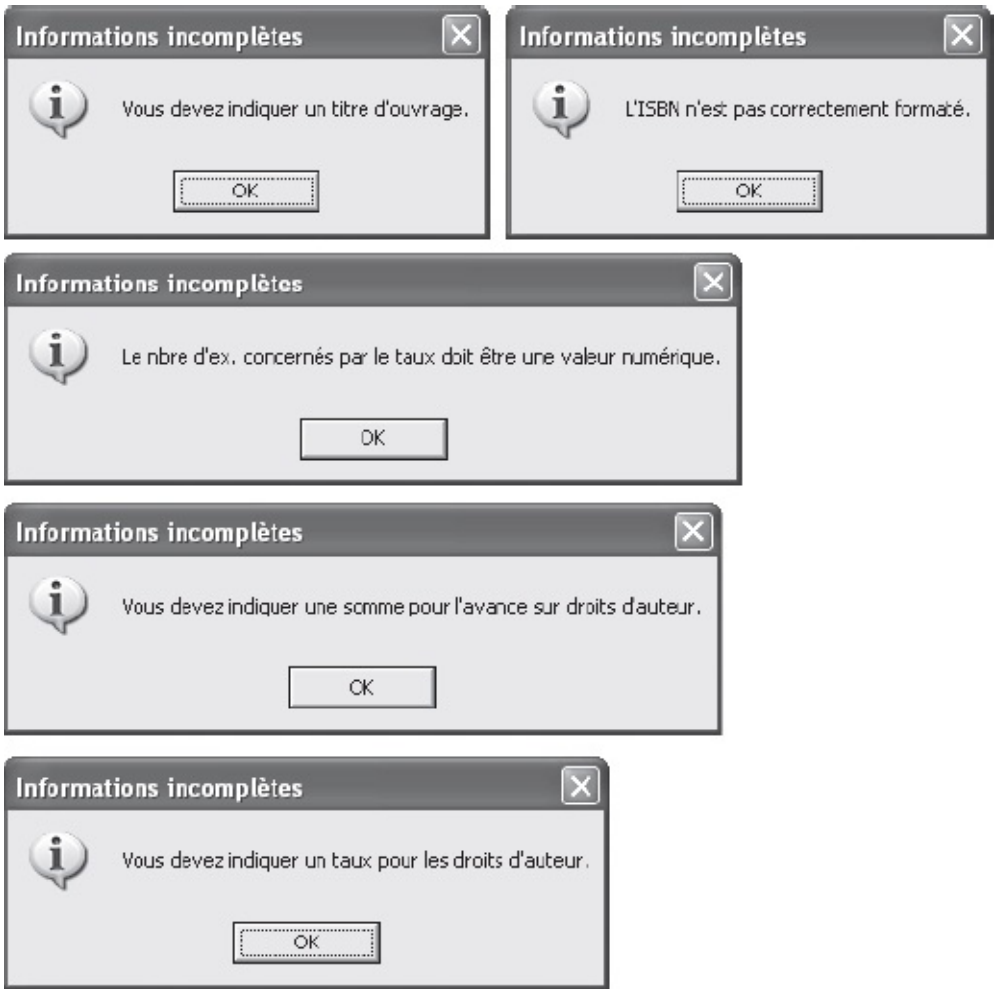


Figure 17-19 – *La validité des données est contrôlée.*

Lignes 191 et 192, `RenvoyerChaîne` est appelée à deux reprises pour retourner les deux valeurs de taux sous forme de chaînes de caractères, pour les besoins du

contrat. Placez cette fonction dans le module `ContratsDivers`. Elle se présente comme suit :

```
Public Function RenvoyerChaîneValeur(valeur)
   Select Case valeur
     Case 0
        RenvoyerChaîneValeur = "zéro"
     Case 1
        RenvoyerChaîneValeur = "un"
     Case 2
        RenvoyerChaîneValeur = "deux"
     Case 3
        RenvoyerChaîneValeur = "trois"
...
     Case 19
        RenvoyerChaîneValeur = "dix-neuf"
     Case 20
        RenvoyerChaîneValeur = "vingt"
   End Select
End Function
```

Conseil

À ce stade, placez l'instruction d'affichage de la feuille `fmContratConditions` dans la procédure `EditionContratAuteur` et exécutez cette dernière afin de vérifier que votre programme ne contient pas d'erreurs. Testez les différentes conditions en entrant des données variables dans les interfaces. `EditionContratAuteur` doit maintenant se présenter comme suit :

```
Sub EditionContratAuteur()
  Set ClasseurAuteurs = GetObject("C:\Mes documents\Classeur Auteurs.xlsx")
  Set MonTableauWord = GetObject(, "Word.Application")
  If Err.Number<>0 Then Err.Clear
  fmContratAuteur.Show
  fmContratConditions.Show
End Sub
```

Feuille fmContratDates

La feuille `fmContratDates` demandera à l'utilisateur d'indiquer les dates de remise et de parution. Nous utiliserons pour cela le contrôle `Calendar`. Une liste modifiable permettra de sélectionner ce que l'utilisateur veut préciser : date de remise ou date de parution. Deux zones de texte dont la fonction sera précisée par deux `Label` montreront les dates sélectionnées sur le calendrier, mais leur valeur ne pourra être modifiée. Enfin, nous retrouverons les mêmes boutons de commande que pour la feuille `fmContratConditions`, à savoir Annuler, Retour et Suite (figure 17-20).

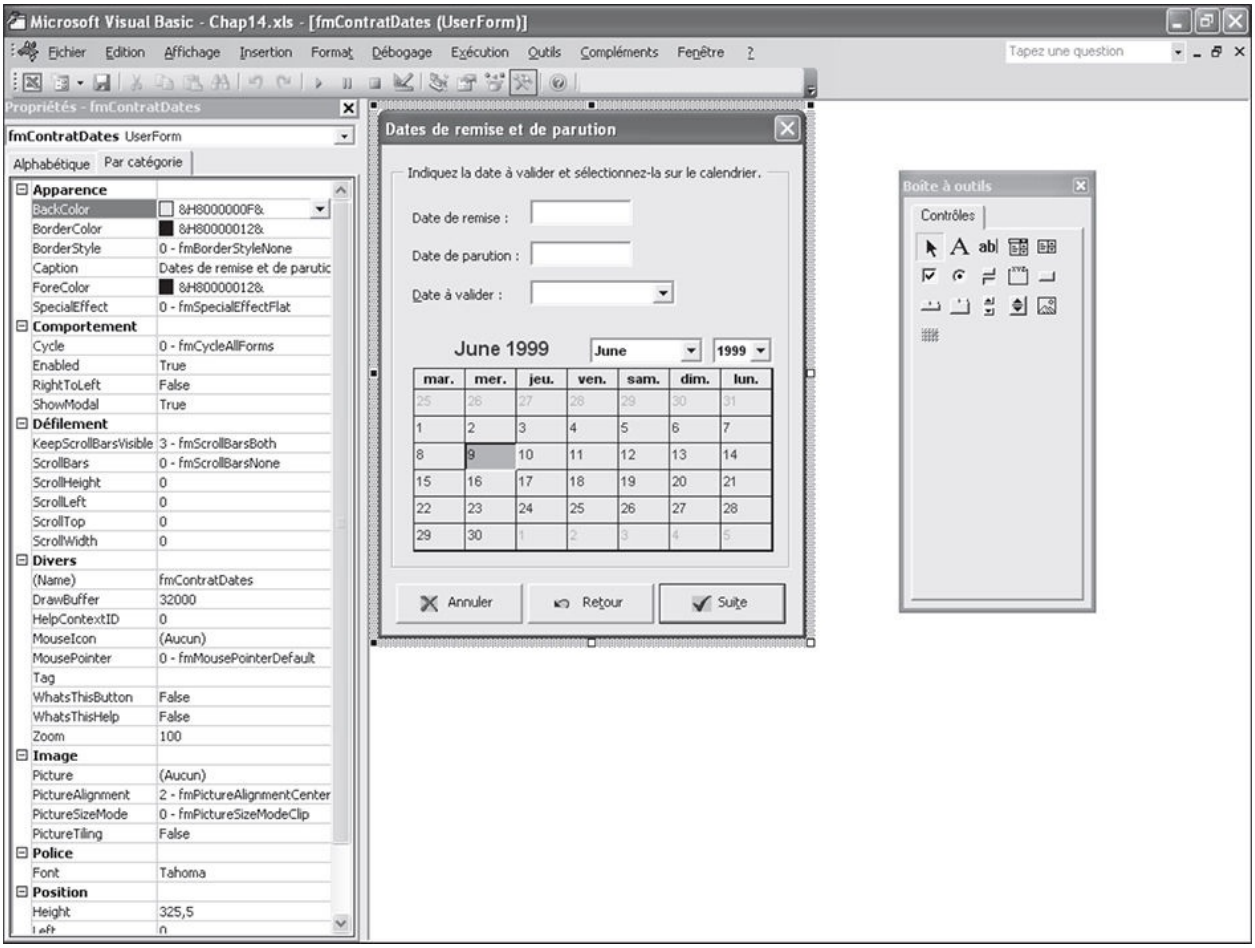


Figure 17-20 – *La feuille fmContratDates en mode Conception.*

`Calendar` ne fait pas partie des contrôles standards de la boîte à outils Visual Basic Editor et n'est pas livré avec toutes les versions d'Excel. Nous l'avons joint aux codes sources des exemples du livre. Commencez par ajouter ce contrôle ; la procédure à suivre est décrite dans la section « Personnaliser la boîte à outils » du chapitres 12.

Pour « dessiner » la feuille `fmContratDates`, commencez par placer un `Frame` contenant deux `TextBox` et un `ComboBox`. Déposez trois `Label` à côté de chacun d'eux afin d'en préciser la fonction. Déposez et dimensionnez le contrôle `Calendar`. Déposez à l'extérieur du cadre les trois boutons de commande.

Astuce

Copiez les boutons de la feuille `fmContratConditions` et collez-les sur `fmContratDates`. Ainsi vous n'aurez pas à en redéfinir les propriétés. Vous pouvez également copier-coller les procédures `cmdAnnuler_Click` et `UserForm_QueryClose` puisqu'elles sont identiques d'une feuille à l'autre.

Définissez les propriétés des contrôles comme indiqué dans le tableau suivant :

Propriété	Valeur
Feuille	
Name	fmContratDates
Caption	Dates de remise et de parution
Contrôle *Frame*	
Caption	Indiquez la date à valider et sélectionnez-la sur le calendrier
Contrôle *TextBox* 1	
Name	txtRemise
Enabled	False
Contrôle *TextBox* 2	
Name	txtParution
Enabled	False
Contrôle *ComboBox*	
Name	cboDateAValider
Style	2 - fmStyleDropDownList
Contrôle *Label* 1	
Caption	Date de remise :
Contrôle *Label* 2	
Caption	Date de parution :
Contrôle *Label* 3	
Caption	Date à valider :
Contrôle *Calendar*	
Name	Calendrier
Contrôle *CommandButton* 1	
Name	cmdAnnuler
Caption	Annuler
Cancel	True
Contrôle *CommandButton* 2	
Name	cmdRetour
Caption	Retour
Contrôle *CommandButton* 3	
Name	cmdSuite
Caption	Suite
Default	True

Ouvrez ensuite la fenêtre Code de la feuille et placez-y les procédures événementielles reproduites ci-après :

```
 1: Private Sub UserForm_Initialize()
 2:   cboDateAValider.AddItem ("Date de remise")
 3:   cboDateAValider.AddItem ("Date de parution")
 4:   cboDateAValider.ListIndex = 0
 5:   Calendrier.Value = Date + 30
 6:   Calendrier.SetFocus
 7: End Sub
 8:
 9: Private Sub Calendrier_Click()
10:   If cboDateAValider.ListIndex=0 Then
11:     txtRemise.Value = Calendrier.Value
12:     cboDateAValider.ListIndex = 1
13:   Else
14:     TxtParution.Value = Calendrier.Value
15:   End If
16: End Sub
17:
18: Private Sub CmdAnnuler_Click()
... Mêmes instructions que pour la feuille fmContratConditions
28: End Sub
29:
30: Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
... Mêmes instructions que pour la feuille fmContratConditions
40: End Sub
41:
42: Private Sub cmdRetour_Click()
43:   fmContratConditions.Show
44: End Sub
45:
46: Private Sub cmdSuite_Click()
47:
48:   'Vérifier la validité des informations
49:   If txtRemise.Value="" Then
50:     MsgBox "Vous devez spécifier une date de remise.", _
51:       vbOKOnly + vbCritical, "Contrat d'auteur"
52:     Exit Sub
53:   ElseIf TxtParution.Value="" Then
54:     MsgBox "Vous devez spécifier une date de parution.", _
55:       vbOKOnly + vbCritical, "Contrat d'auteur"
56:     Exit Sub
57:   ElseIf DateValue(TxtParution)-DateValue(txtRemise)<=0 Then
58:     MsgBox "Vous devez spécifier une date de remise antérieure à la date de
        ➥ parution.", _
59:       vbOKOnly + vbCritical, "Contrat d'auteur"
60:     Exit Sub
61:   ElseIf DateValue(TxtParution)-DateValue(txtRemise)<40 Then
62:     Dim continuer As Integer
63:     continuer = MsgBox("Vous avez indiqué une date de remise à seulement " _
64:       & DateValue(TxtParution) - DateValue(txtRemise) _
65:       & " jours de la date de parution. Valider cette date ?",
        ➥ vbYesNo + vbQuestion, _
66:       "Valider la date de remise ?")
67:     If continuer=vbNo Then Exit Sub
68:   End If
69:
70:   Me.Hide
71:
72:   'Validation des dates
73:   MesConditionsAuteur.remise = txtRemise.Value
74:   MesConditionsAuteur.Parution = TxtParution.Value
75: End Sub
```

La procédure d'initialisation (lignes 1 à 7) ajoute les entrées Date de remise et

Date de parution à la liste modifiable (lignes 2 et 3), puis active le premier élément de la liste (ligne 4). Ligne 5 et 6, la valeur du calendrier est définie à 30 jours de la date courante – renvoyée par la fonction `Date` – et le calendrier reçoit le focus.

`Calendrier_Click` (lignes 9 à 16) gère l'affectation des dates aux zones de texte `txtRemise` et `txtParution`. Une structure conditionnelle est utilisée pour définir l'entrée choisie dans la liste modifiable et écrire la date retenue dans la zone de texte correspondante (lignes 11 et 14). Si le premier élément de la liste est sélectionné (Date de remise), `txtRemise` reçoit la valeur, puis le second élément est sélectionné (ligne 12), ce qui permet à l'utilisateur de choisir immédiatement la date de parution.

`cmdRetour_Click` (lignes 42 à 44) gère les clics sur le bouton Retour. Elle affiche la feuille `fmContratConditions`.

Enfin, `cmdSuite_Click` (lignes 46 à 75) vérifie la validité des dates indiquées (lignes 48 à 68) et affecte ces valeurs aux variables appropriées. Une seule instruction conditionnelle est utilisée. Lignes 49 à 56, on vérifie que les deux dates ont été précisées. Si tel n'est pas le cas, un message s'affiche à l'attention de l'utilisateur et une instruction `Exit` entraîne la sortie de la procédure (voir figure 17-21).

Figure 17-21 – *Les dates de remise et de parution doivent être précisées.*

Lignes 57 à 60, on vérifie que la date de parution est postérieure à celle de remise, grâce à la fonction `DateValue` qui renvoie la valeur de la date sous forme numérique (consultez l'aide en ligne pour plus de précisions). Si tel n'est pas

le cas, un message s'affiche à l'attention de l'utilisateur et une instruction `Exit` entraîne la sortie de la procédure. Lignes 61 à 68, on vérifie si plus de 40 jours séparent les deux dates. Si tel n'est pas le cas, l'utilisateur en est averti par un message. Il peut alors valider les dates ou cliquer sur le bouton Non, ce qui entraîne la sortie de la procédure (ligne 67).

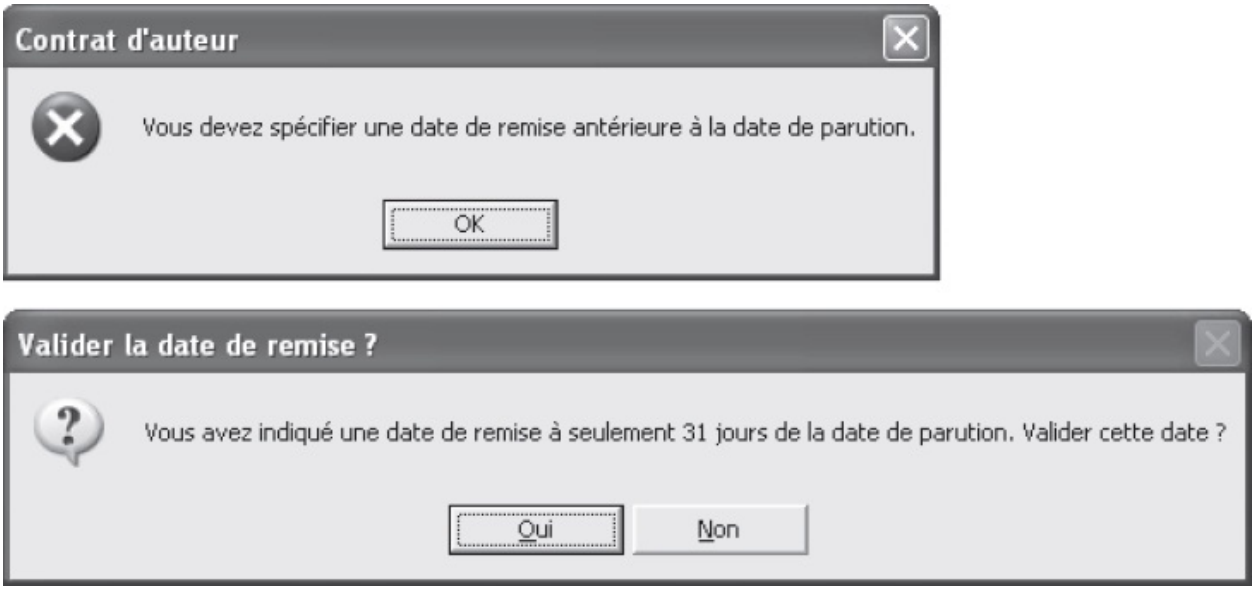

Contrat d'auteur
Vous devez spécifier une date de remise antérieure à la date de parution.
OK
Valider la date de remise ?
Vous avez indiqué une date de remise à seulement 31 jours de la date de parution. Valider cette date ?
Oui
Non

Figure 17-22 – *Les dates proposées sont comparées à l'aide de la fonction DateValue.*

Ligne 70, la feuille est masquée. Les dates fournies sont ensuite affectées aux espaces de stockage appropriés de la variable `MesConditionsAuteur`.

Conseil

Placez l'instruction d'affichage de la feuille `fmContratDates` dans la procédure `EditionContratAuteur` et exécutez cette dernière afin de vérifier que votre programme ne contient pas d'erreurs. `EditionContratAuteur` doit maintenant se présenter comme suit :

```
Sub EditionContratAuteur()
  Set ClasseurAuteurs = GetObject("C:\Mes documents\Classeur Auteurs.xlsx")
  Set MonTableauWord = GetObject(, "Word.Application")
  If Err.Number<>0 Then Err.Clear
  fmContratAuteur.Show
  fmContratConditions.Show
  fmContratDates.Show
End Sub
```

Feuille fmContratImpression

Sur la feuille `fmContratImpression` (figure 17-23), une case à cocher demandera à l'utilisateur s'il souhaite imprimer. Si oui, deux autres cases à cocher s'afficheront pour qu'il indique si les feuilles de paie, le contrat ou les deux doivent être imprimés. Une zone de texte et un bouton toupie serviront à préciser le nombre de copies souhaitées pour chaque impression. Enfin, nous retrouverons les trois boutons de retour, d'annulation et de validation des feuilles précédemment créées.

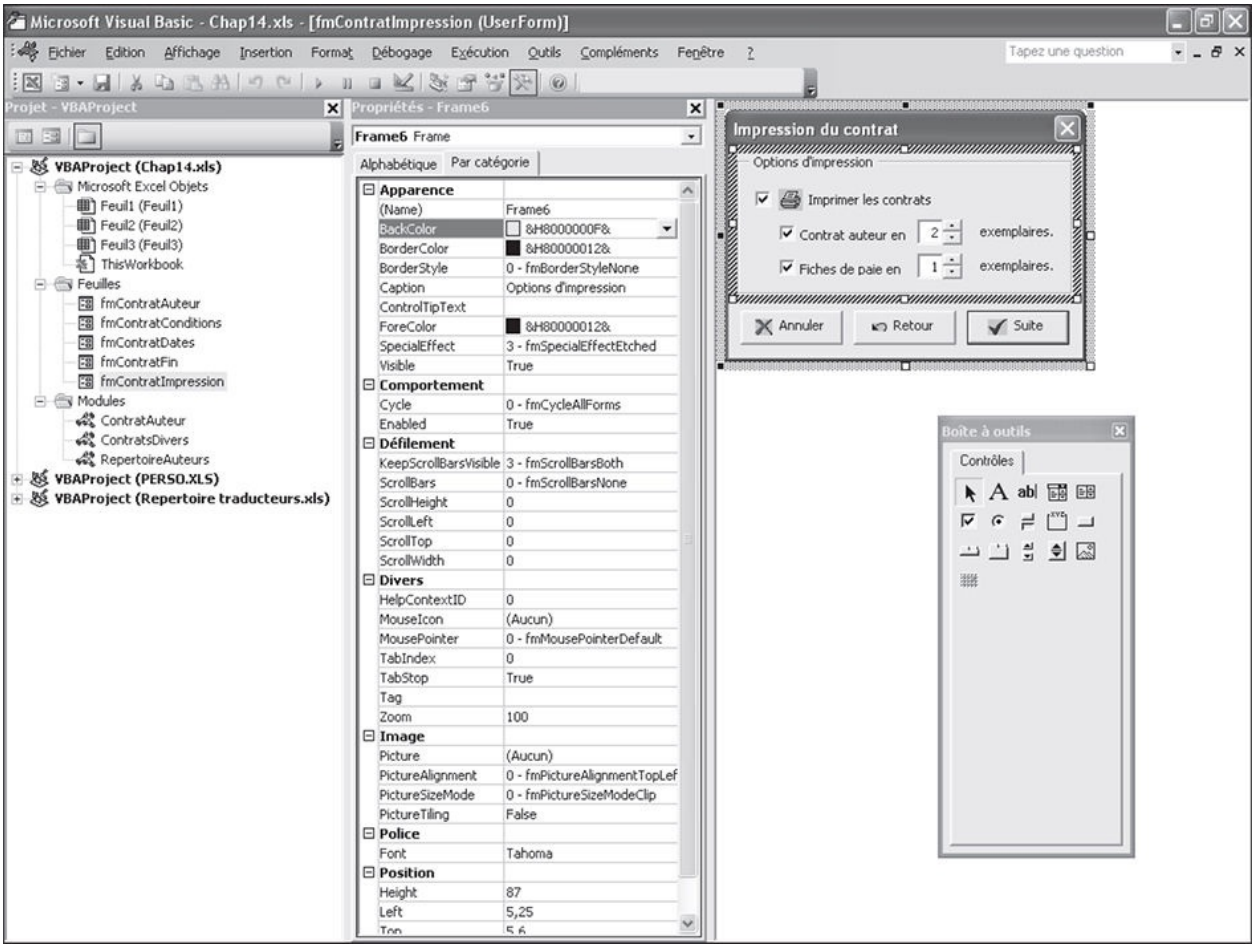


Figure 17-23 – *La feuille fmContratImpression en mode Conception.*

Astuce

Notez l'image de l'imprimante apparaissant entre la première case à cocher et son libellé. Il s'agit d'un contrôle `Picture` qui a été placé sur le `CheckBox`. Nous avons ajouté des espaces devant le texte de la propriété `Caption` de ce dernier afin de ne pas masquer le libellé de la case à cocher.

Commencez par placer un contrôle `Frame` sur la feuille, dans lequel vous ajouterez trois `CheckBox` (en plaçant les deux derniers en retrait par rapport au

premier), puis un `TextBox`, un `SpinButton` et un `Label` à côté de chaque case à cocher. Enfin, placez trois boutons de commande hors du cadre. Définissez les propriétés des contrôles comme indiqué dans le tableau suivant.

Propriété	Valeur
Feuille	
`Name`	`fmContratImpression`
`Caption`	Impression du contrat
Contrôle *`Frame`*	
`Caption`	Options d'impression
Contrôle *`CheckBox`* 1	
`Name`	`chkImprimer`
`Caption`	Imprimer les contrats
`Value`	`True`
Contrôle *`CheckBox`* 2	
`Name`	`chkImpressionContrat`
`Caption`	Contrat auteur en
`Value`	`True`
Contrôle *`CheckBox`* 3	
`Name`	`chkImpressionPaie`
`Caption`	Fiches de paie en
`Value`	`True`
Contrôle *`TextBox`* 1	
`Name`	`txtQteContrat`
`Value`	2
Contrôle *`TextBox`* 2	
`Name`	`txtQtePaie`
`Value`	1
Contrôle *`SpinButton`* 1	
`Name`	`spinContrat`
`Value`	2
`Min`	1
`Max`	10
Contrôle *`SpinButton`* 2	
`Name`	`spinPaie`
`Value`	1
`Min`	1
`Max`	10
Contrôles *`Label`* 1 et 2	

Caption	Exemplaire(s)
Contrôle *CommandButton* 1	
Name	cmdAnnuler
Caption	Annuler
Cancel	True
Contrôle *CommandButton* 2	
Name	cmdRetour
Caption	Retour
Contrôle *CommandButton* 3	
Name	cmdTerminer
Caption	Terminer
Default	True

Ouvrez la fenêtre Code de la feuille et placez-y les procédures événementielles suivantes :

```
 1: Private Sub UserForm_Initialize()
 2:   If MonAuteur.societe=True Then
 3:     chkImpressionPaie.Value = False
 4:     chkImpressionPaie.Enabled = False
 5:     txtQtePaie.Enabled = False
 6:     spinPaie.Enabled = False
 7:   End If
 8: End Sub
 9:
10: Private Sub chkImprimer_Click()
11:   If chkImprimer.Value=False Then
12:     chkImpressionPaie.Enabled = False
13:     chkImpressionContrat.Enabled = False
14:     txtQteContrat.Locked = True
15:     txtQtePaie.Locked = True
16:     spinPaie.Enabled = False
17:     spinContrat.Enabled = False
18:   Else
19:     If MonAuteur.societe=False Then
20:       chkImpressionPaie.Enabled = True
21:       txtQtePaie.Locked = False
22:       spinPaie.Enabled = True
23:     End If
24:     chkImpressionPaie.Value = True
25:     chkImpressionContrat.Enabled = True
26:     chkImpressionContrat.Value = True
27:     txtQteContrat.Locked = False
28:     spinContrat.Enabled = True
29:   End If
30: End Sub
31:
32: Private Sub chkImpressionContrat_Click()
33:   If chkImpressionContrat.Value=False Then
34:     txtQteContrat.Locked = True
35:     spinContrat.Enabled = False
36:   Else
37:     txtQteContrat.Locked = False
38:     spinContrat.Enabled = True
39:   End If
```

```
 40: End Sub
 41:
 42: Private Sub chkImpressionPaie_Click()
 43:   If chkImpressionPaie.Value=False Then
 44:     txtQtePaie.Locked = True
 45:     spinPaie.Enabled = False
 46:   Else
 47:     txtQtePaie.Locked = False
 48:     spinPaie.Enabled = True
 49:   End If
 50: End Sub
 51:
 52: Private Sub spinContrat_Change()
 53:   txtQteContrat.Value = spinContrat.Value
 54: End Sub
 55:
 56: Private Sub spinPaie_Change()
 57:   txtQtePaie.Value = spinPaie.Value
 58: End Sub
 59:
 60: Private Sub txtQteContrat_Change()
 61:   If IsNumeric(txtQteContrat.Value)=False Then
 62:     MsgBox "Le nombre de copies doit être une valeur numérique.", _
 63:       vbOKOnly + vbInformation, "Informations erronées"
 64:     txtQteContrat.Value = ""
 65:     txtQteContrat.SetFocus
 66:     Exit Sub
 67:   ElseIf txtQteContrat.Value<1 Or txtQteContrat.Value>10 Then
 68:     MsgBox "Le nombre de copies doit être compris entre 1 et 10.", _
 69:       vbOKOnly + vbInformation, "Valeur non valide"
 70:     txtQteContrat.Value = ""
 71:     txtQteContrat.SetFocus
 72:     Exit Sub
 73:   End If
 74:   spinContrat.Value = txtQteContrat.Value
 75: End Sub
 76:
 77: Private Sub txtQtePaie_Change()
 78:   If IsNumeric(txtQtePaie.Value)=False Then
 79:     MsgBox "Le nombre de copies doit être une valeur numérique.", _
 80:       vbOKOnly + vbInformation, "Informations erronées"
 81:     txtQtePaie.Value = ""
 82:     txtQtePaie.SetFocus
 83:     Exit Sub
 84:   ElseIf txtQtePaie.Value<1 Or txtQtePaie.Value>10 Then
 85:     MsgBox "Le nombre de copies doit être compris entre 1 et 10.", _
 86:       vbOKOnly + vbInformation, "Valeur non valide"
 87:     txtQtePaie.Value = ""
 88:     txtQtePaie.SetFocus
 89:     Exit Sub
 90:   End If
 91:   spinPaie.Value = txtQtePaie.Value
 92: End Sub
 93:
 94: Private Sub cmdAnnuler_Click()
... Mêmes instructions que pour la feuille fmContratCondition
101: End Sub
102:
103: Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
... Mêmes instructions que pour la feuille fmContratCondition
112: End Sub
113:
114: Private Sub cmdRetour_Click()
115:   fmContratConditions.Show
```

```
116: End Sub
117:
118: Private Sub cmdSuite_Click()
119:   With MonImpression
120:     .Imprimer = chkImprimer.Value
121:     .ImprimerCourrier = chkImpressionContrat.Value
122:     .ImprimerPaie = chkImpressionPaie.Value
123:     .nbreCourrier = txtQteContrat.Value
124:     .nbrePaie = txtQtePaie.Value
125:   End With
126:   Me.Hide
127: End Sub
```

La procédure d'initialisation de la feuille (lignes 1 à 8) utilise une structure conditionnelle pour désactiver les contrôles dédiés à l'impression des feuilles de paie, si le contrat est édité au nom d'une société – les sociétés facturent tandis qu'on établit une feuille de paie au nom des personnes physiques. On teste pour cela la variable `MonAuteur.Societe` qui a reçu la valeur `True` ou `False` lorsque l'utilisateur a validé la feuille `fmContratAuteur`. Si la valeur est `True`, `chkImpressionPaie` est décochée (ligne 3), puis rendue inaccessible (ligne 4). La zone de texte et le bouton toupie associés sont également rendus inaccessibles (lignes 5 et 6).

`chkImprimer_Click` modifie l'état des contrôles définissant les éléments à imprimer et le nombre de copies à réaliser. Si le bouton est décoché, les deux autres cases sont désactivées (lignes 12 et 13), les deux zones de texte correspondantes sont verrouillées (lignes 14 et 15) et les deux boutons toupies associés à ces dernières rendus inaccessibles (lignes 16 et 17). Dans le cas contraire, les instructions des lignes 19 à 29 s'exécutent. Lignes 19 à 24, une structure conditionnelle imbriquée active les contrôles définissant le nombre d'impression des feuilles de paie, si le contrat n'est pas établi au nom d'une société. Les contrôles servant à définir l'impression du contrat sont activés (lignes 25 à 28).

Lignes 32 à 40, `chkImpressionContrat_Click` active ou désactive l'accès à la zone de texte et au bouton toupie correspondant à l'impression du contrat. Lignes 42 à 50, `chkImpressionPaie_Click` effectue les mêmes opérations pour la case à cocher libellée Fiches de paie.

Les deux procédures des lignes 52 à 58 affectent la valeur du bouton toupie à la case correspondante lorsque l'utilisateur clique sur l'un des contrôles `spinContrat` ou `spinPaie`.

Lignes 60 à 75 et lignes 77 à 92, les procédures `txtQteContrat_Change` et `txtQtePaie_Change` sont identiques aux procédures `txtTaux1_Change` et `txtTaux2_Change` de la feuille `fmContratConditions`. Elles valident les valeurs saisies dans les zones de texte et les affectent aux contrôles `SpinButton` correspondants.

La procédure `cmdRetour_Click` (lignes 114 à 116) affiche la feuille `fmContratDates`.

Enfin, `cmdSuite_Click` (lignes 118 à 127) transfère les informations fournies dans la variable `MonImpression`.

Feuille fmContratFin

La feuille `fmContratFin` est très simple. Elle indique à l'utilisateur que les informations nécessaires ont été fournies et que le contrat va être édité. Elle contient un seul bouton et un contrôle `Label`. Réalisez cette feuille en vous appuyant sur sa représentation de la figure 17-24 (nous avons affecté une image bitmap à la propriété `Picture` du contrôle). Définissez la propriété `Caption` du bouton de commande à `cmdOK`.

Figure 17-24 – *Toutes les informations nécessaires sont maintenant stockées dans des variables*

Placez maintenant les instructions suivantes dans la fenêtre Code de la feuille `fmContratFin` :

```
Private Sub cmdOK_Click()
   Me.Hide
End Sub


Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
   Dim rep As Byte
   rep = MsgBox("Etes-vous sûr de vouloir annuler l'édition du contrat en cours ?",_
     vbYesNo + vbQuestion, "Annuler l'édition de contrat ?")
   If rep=vbNo Then
     Exit Sub
   End If
   Me.Hide
   End
End Sub
```

À ce stade du développement, la procédure `EditionContratAuteur` doit se présenter comme suit :

```
Sub EditionContratAuteur()
   '1. Liaisons des variables objets
   Set ClasseurAuteurs = GetObject("C:\Mes documents\Classeur Auteurs.xlsx")
   Set MonTableauWord = GetObject(, "Word.Application")
   If Err.Number<>0 Then Err.Clear
```

```
   ’2. Affichage des feuilles
   fmContratAuteur.Show
   fmContratConditions.Show
   fmContratDates.Show
   fmContratImpression.Show
   fmContratFin.Show


   ’3. Edition des feuilles de paie et du courrier
   ’4. Mise à jour du tableau Word
   ’5. Libération des ressources mémoire
End Sub
```

Écriture des procédures d’édition de documents

Les interfaces sont maintenant développées, les données stockées dans des variables. Il ne nous reste plus qu’à écrire les procédures qui établiront le contrat et les feuilles de paie, les imprimeront et les enregistreront, puis mettront à jour le tableau Word.

Édition des feuilles de paie

Commencez par modifier la procédure `EditionContratAuteur` en y insérant l’appel de `EditerFeuilleDePaie`, qui ne s’exécute que si le contrat n’est pas édité au nom d’une société :

```
Sub EditionContratAuteur()
   ’1. Liaisons des variables objets
   Set ClasseurAuteurs = GetObject("C:\Mes documents\Classeur Auteurs.xlsx")
   Set MonTableauWord = GetObject(, "Word.Application")
   If Err.Number<>0 Then Err.Clear


   ’2. Affichage des feuilles
   fmContratAuteur.Show
   fmContratConditions.Show
   fmContratDates.Show
   fmContratImpression.Show
   fmContratFin.Show


   ’3. Edition des feuilles de paie et du courrier
   If MonAuteur.societe=False Then
      Call EditerFeuilleDePaie
   End If


   ’4. Mise à jour du tableau Word
   ’5. Libération des ressources mémoire
End Sub
```

Placez maintenant la procédure `EditerFeuilleDePaie` dans le module `ContratAuteur` :

```
 1: Sub EditerFeuilleDePaie()
 2:   Dim feuillnum As Byte
 3:
```

```
  For feuillnum = 1 To MesConditionsAuteur.NumPaiements
    'Edition de la (des) feuille(s)
    Workbooks.Add Template:="C:\Documents and settings\Administrateur\
     ➥ Application Data\Microsoft\Modèles\PAIEMENT.xlt"
    With ActiveWorkbook.Sheets(1)
      If MonAuteur.societe=False Then
        .Range("C2").Value = MonAuteur.Prenom & " " & MonAuteur.Nom
      Else
        .Range("C2").Value = MonAuteur.Nom
      End If

      .Range("C3").Value = MesConditionsAuteur.Titre
      .Range("C4").Value = MesConditionsAuteur.ISBN
      .Range("C6").Value = MesConditionsAuteur.remise
      .Range("C7").Value = MesConditionsAuteur.Parution
      .Range("C9").Value = MonAuteur.Adresse
      .Range("C10").Value = MonAuteur.CodePostal
      .Range("C11").Value = MonAuteur.Ville
      .Range("C12").Value = MonAuteur.Pays
      .Range("F15").Value = _
        MesConditionsAuteur.AvanceSurDroits / MesConditionsAuteur.NumPaiements

      If MesConditionsAuteur.NumPaiements=1 Then
        .Range("F30").Value = MesConditionsAuteur.Parution
      Else
        If feuillnum=1 Then
          .Range("F30").Value = MesConditionsAuteur.remise
        Else
          .Range("F30").Value = MesConditionsAuteur.Parution
        End If
      End If
      If MesConditionsAuteur.TauxVariable=True Then
        .Range("B32") = "Pour les " & _
          MesConditionsAuteur.NumExemplairesTaux1 & _
          " premiers ouvrages"
        .Range("B33") = "Au-delà"
        .Range("F32") = MesConditionsAuteur.TauxDroitsNum1 & " %"
        .Range("G32") = MesConditionsAuteur.TauxDroitsStr1 & " pour cent"
        .Range("F33") = MesConditionsAuteur.TauxDroitsNum2 & " %"
        .Range("G33") = MesConditionsAuteur.TauxDroitsStr2 & " pour cent"
      Else
        .Range("B32") = ""
        .Range("B33") = ""
        .Range("F32") = MesConditionsAuteur.TauxDroitsNum1 & " %"
        .Range("G32") = MesConditionsAuteur.TauxDroitsStr1 & " pour cent"
        .Range("F33") = ""
        .Range("G33") = ""
      End If
    End With
    Application.Calculate

    'impression du classeur
    If MonImpression.Imprimer=True And MonImpression.ImprimerPaie=True
     ➥ Then_
      ActiveWorkbook.PrintOut copies:=MonImpression.nbrePaie
    End If

    'enregistrement du classeur
    ActiveWorkbook.SaveAs Filename:="C:\Mes Documents\" & _
      MesConditionsAuteur.ISBN & " " & MonAuteur.Nom & " " & _
      "Paie" & feuillnum & ".xlsx", FileFormat:=xlNormal
    ActiveWorkbook.Close
  Next feuillnum
End Sub
```

Cette procédure utilise une boucle `For…Next` pour éditer le nombre de feuilles de paie approprié, soit `MesConditionsAuteur.NumPaiements`.

À chaque passage de la boucle, un classeur fondé sur le modèle PAIEMENT.xlt est créé (ligne 6). Penser à adapter le chemin apparaissant dans ce listing, qui est celui du dossier de stockage des modèles. Lignes 7 à 34, une structure `With… End With` est utilisée pour insérer les données appropriées dans les cellules du classeur. Ligne 22, l'avance sur droits d'auteur est divisée par le nombre de paiements. Lignes 25 à 33, une structure conditionnelle sert à définir les dates de règlement : si un seul paiement est effectué, le règlement s'opère à la parution (ligne 26), si deux versements sont demandés (lignes 27 à 33), le premier est versé lors de la remise (ligne 29) et le second lors de la parution (ligne 31). Lignes 35 à 50, une structure conditionnelle complète les informations concernant les taux des droits d'auteur. Ligne 52, le calcul est forcé. Lignes 55 à 57, le classeur est imprimé. Les lignes 60 à 62 correspondent à une seule instruction d'enregistrement. Le classeur est enregistré dans le dossier C:\Mes documents et a pour nom *ISBN Nom* Paie1.xlsx ou *ISBN Nom* Paie2.xlsx. Ligne 63, le classeur est fermé.

Mise à jour du tableau Word

Commencez par insérer l'appel de la procédure de mise à jour du tableau dans `EditionContratAuteur` :

```
Sub EditionContratAuteur()
   ’1. Liaisons des variables objets
   Set ClasseurAuteurs = GetObject("C:\Mes documents\Classeur Auteurs.xlsx")
   Set MonTableauWord = GetObject(, "Word.Application")
   If Err.Number<>0 Then Err.Clear


   ’2. Affichage des feuilles
   fmContratAuteur.Show
   fmContratConditions.Show
   fmContratDates.Show
   fmContratImpression.Show
   fmContratFin.Show


   ’3. Edition des feuilles de paie et du courrier
   If MonAuteur.societe=False Then
      Call EditerFeuilleDePaie
   End If
   Call MettreTableauAJour


   ’4. Mise à jour du tableau Word
   ’5. Libération des ressources mémoire
End Sub
```

Placez ensuite le code suivant dans le module `ContratsAuteur` :

```
 1: Sub MettreTableauAJour()
 2:   Dim derligne As Integer
 3:   Set MonTableauWord = GetObject("C:\Mes documents\Contrats Auteurs.doc")
 4:   MonTableauWord.Tables(1).Rows.Add
 5:   derligne = MonTableauWord.Tables(1).Rows.Count
 6:   With MonTableauWord.Tables(1)
 7:     .Cell(derligne,1).Range.InsertAfter MesConditionsAuteur.ISBN
 8:     .Cell(derligne,2).Range.InsertAfter MesConditionsAuteur.Titre
 9:     .Cell(derligne,3).Range.InsertAfter MonAuteur.Nom
10:     If MesConditionsAuteur.TauxVariable=True Then
11:       .Cell(derligne,4).Range.InsertAfter MesConditionsAuteur.
          ➥ TauxDroitsNum1 & _
12:         " sur " & MesConditionsAuteur.NumExemplairesTaux1 & _
13:         " puis " & MesConditionsAuteur.TauxDroitsNum2
14:     Else
15:       .Cell(derligne,4).Range.InsertAfter MesConditionsAuteur.TauxDroitsNum1
16:     End If
17:     .Cell(derligne,5).Range.InsertAfter MesConditionsAuteur.AvanceSurDroits
18:     .Cell(derligne,6).Range.InsertAfter Date
19:     .Cell(derligne,7).Range.InsertAfter MesConditionsAuteur.remise
20:     .Cell(derligne,8).Range.InsertAfter MesConditionsAuteur.Parution
21:   End With
22:   MonTableauWord.Save
23:   MonTableauWord.Close
24: End Sub
```

Ligne 3, la variable `MonTableauWord` se voit affecter un document Word. Lignes 4 et 5, une ligne est ajoutée au premier tableau de ce document et son numéro est stocké dans la variable `DerLigne`. Lignes 6 à 21, une structure `With…End With` complète les cellules du tableau avec les valeurs appropriées. Lignes 22 et 23, le tableau est enregistré puis fermé.

Telle qu'elle se présente, cette application fonctionne. Pour la sécuriser, vous devez cependant la tester dans des conditions différentes et mettre en place des gestionnaires d'erreur. De multiples conditions peuvent causer une erreur. Ce sera par exemple le cas si un document utilisé (le modèle PAIEMENT.xlt ou le document Contrats Auteur.doc par exemple) n'est pas disponible, ou est en cours d'utilisation par un autre utilisateur.

Annexe

Mots-clés pour la manipulation de fichiers et de dossiers

Cette annexe présente les mots-clés dédiés à la gestion de fichiers et de dossiers. Manipulez-les avec prudence afin de ne pas supprimer malencontreusement des données importantes !

Mots-clés pour la manipulation des fichiers et des dossiers

Mot-clé	Description	Exemple
`ChDir` *`path`*	Change de dossier. Notez que le lecteur courant n'est pas modifié.	`ChDir "C:\Mes documents"`. Fait de C:\Mes documents le dossier actif.
`ChDrive` *`drive`*	Change de lecteur.	`ChDrive "D:"` Fait de D: le lecteur actif.
`CurDir`	Retourne le chemin du dossier en cours.	
`Dir(`*`pathname`*`,` *`attributes`*`)`	Retourne le nom des fichiers d'un dossier ou d'un volume. *`pathname`* est facultatif et peut représenter le nom du dossier et le lecteur. *`attributes`* est facultatif et représente les attributs des fichiers que l'on veut retourner. Additionnez les valeurs suivantes pour définir *`attributes`* : – `vbNormal` ou 0 : Fichiers sans attributs (valeur par défaut). – `vbReadOnly` ou 1 : Fichiers accessibles en lecture seule et fichiers sans attributs. – `vbHidden` ou 2 : Fichiers cachés et fichiers sans attributs. – `vbSystem` ou 4 : Fichiers système et fichiers sans attributs (non disponible sur Macintosh). – `vbVolume` ou 8 : Nom de volume ; si un autre attribut valeur est précisé, la constante `vbVolume` est ignorée (non disponible sur Macintosh). – `vbDirectory` ou 16 : Dossiers et fichiers sans attributs.	`MyFile = Dir("*.doc",` `vbHidden)`. Retourne le premier fichier possédant l'extension .doc et l'attribut "Fichier caché".
	Copie un fichier *`source`* à l'emplacement	`FileCopy "C:\toto.doc",`

`FileCopy source, destination`	*destination*. Notez que le nom du fichier copié peut être différent de celui du fichier source. Attention : une erreur est générée si vous tentez de copier un fichier ouvert.	`"A:infos.doc"` Copie le fichier toto.doc sur le lecteur a: et sous le nom infos.doc.
`FileDateTime (pathname)`	Retourne les informations de date et d'heure d'un fichier.	`FileDateTime("C:\toto.doc")` Retourne la date et l'heure du fichier toto.doc.
`FileLen(pathname)`	Retourne la taille d'un fichier en octets.	`FileLen("C:\toto.doc")` Retourne la taille de toto.doc.
`FullName`	Retourne le nom complet d'un fichier (avec son chemin).	`ActiveDocument.FullName` Retourne le nom complet du document actif.
`GetAttr(pathname)`	Renvoie une valeur représentant les attributs d'un fichier, d'un dossier ou d'un volume, qui résulte de l'addition des valeurs suivantes : – `vbNormal` ou 0 : Fichier normal. – `vbReadOnly` ou 1 : Fichier en lecture seule. – `vbHidden` ou 2 : Fichier caché. – `vbSystem` ou 4 : Fichier système (non disponible sur Macintosh). – `vbDirectory` ou 16 : Dossier. – `vbArchive` ou 32 : Fichier modifié depuis la dernière sauvegarde (non disponible sur Macintosh).	
`MkDir path`	Crée un nouveau dossier.	`MkDir "C:\MonDossier"` Crée le dossier C:\MonDossier.
`Name`	Retourne le nom d'un fichier (sans son chemin).	`ActiveDocument.Name` Retourne le nom du document actif.
`RmDir Path`	Supprime un dossier.	`RmDir "C:\MonDossier"` Supprime le dossier C:\MonDossier.
`SetAttr pathname, attributes`	Modifie les attributs d'un fichier. Utilisez les mêmes valeurs que pour la fonction `GetAttr()` pour définir les attributs du fichier. Si le fichier est ouvert, une erreur est générée.	`SetAttr "C:\toto.doc", vbHidden + vbReadOnly` Affecte les attributs Fichier caché et Lecture seule au fichier toto.doc.

Index

Symboles

-, opérateur arithmétique, 156
*, opérateur arithmétique, 156
/, opérateur arithmétique, 156
&, opérateur de concaténation, 150
+, opérateur arithmétique, 156
<, opérateur relationnel, 178
<=, opérateur relationnel, 178
<>, opérateur relationnel, 178
=, opérateur relationnel, 178
>, opérateur relationnel, 178
>=, opérateur relationnel, 178

A

Abs, fonction, 223
Accelerator, propriété, 324
Activate, événement, 376
Activation (feuille de calcul), 376
ActiveCell, propriété, 59, 372
ActiveSheet, propriété, 372
ActiveWorkbook, propriété, 372
afficher
  barre d'outils, 106
  boîte à outils, 89
  Explorateur, 77, 80
  fenêtre Code, 89
  fenêtre Proriétés, 102
  fenêtre UserForm, 89
  feuille, 298
AfterUpdate, événement, 343
aide
  code, 146

VBA, 83, 302
ajouter
module, 129
procédure, 133
Aligner, commande, 289
Alignment, propriété, 303
ancrage, 108
And, opérateur logique, 213
annuler, 373
appels
pile (des), 250, 254
procédures, 114, 126, 128, 140
Application
hôte, 4, 15, 73
objet, 17, 369, 370, 372, 373
procédures (de niveau), 371
arguments
facultatifs, 120
fonctions, 27
nommés, 143
passage (de), 119, 142
passés, 114
Arrêt
enregistrement, 58
mode, 249
As, mot-clé, 120
Asc, fonction, 224
Atn, fonction, 223
automation, 16, 174
AutoSize, propriété, 311
AutoTab, propriété, 312
AutoWordSelect, propriété, 313

B

BackColor, propriété, 304
BackStyle, propriété, 304
barre d’état, 372
barre d’outils
afficher/masquer, 106

Visual Basic Editor, 105
barre de formule, 372
BeforeDoubleClick, événement, 376
BeforeDragOver, événement, 343
BeforeDropOrPaste, événement, 343
BeforeRightClick, événement, 376
BeforeUpdate, événement, 343
bibliothèque d'objets, 16, 80
Excel, 82
MSForms, 81
Office, 81
référencer, 174
VBA, 82
boîte à outils, 76, 276
afficher, 89
ajouter, 293
ajouter une page, 296
contrôle, 281, 282
personnaliser, 293
boîtes de dialogue
afficher, 198, 199, 202, 204
Enregistrer, 42, 209
Excel, 209
Macro, 42
Ouvrir, 209
Boolean, type, 22
BorderColor, propriété, 305
BorderStyle, propriété, 305
boucles, 177
Do Loop, 181
For Each…Next, 189
For…Next, 185, 188
While…Wend, 178
bouton bascule, 280
bouton d'option, 280
bouton de commande, 280
bouton toupie, 283
boutons
macro, 42, 268

Ne pas commenter, 137
réorganiser, 292
ByRef, mot-clé, 120
Byte, type de donnée numérique, 155
ByVal, mot-clé, 120, 121

C

cadre, 278
calcul forcé, 373
feuille de calcul, 376
Calculate
événement, 376
méthode, 373
Call, mot-clé, 141
Cancel, propriété, 313
Caption, propriété, 305, 372
caractère
continuité de ligne, 135
séparation, 372
case à cocher, 279
casse, 45
CBool, fonction, 171
CByte, fonction, 171
CCur, fonction, 171
CDate, fonction, 171
CDbl, fonction, 171
CDec, fonction, 171
Cells, propriété, 59
cellule
active, 57, 372
modification, 376
centrer (contrôles), 291
chaîne de caractères
comparer, 236
concaténer, 229, 231
espaces, 233
extraire, 233
longueur, 233
manipuler, 229

modifier, 229, 235
rechercher, 237
remplacer, 234
type de données, 21
variables, 153
Change, événement, 344, 376
ChDir, instruction, 447
ChDrive, instruction, 447
CheckBox, contrôle, 279, 316, 360
valeur, 308
Chr, fonction, 200, 224, 231
CInt, fonction, 171
classes, 14, 82
membres, 80, 82
classeur, 19, 372, 374
actif, 372
calcul forcé, 373
macros, 47
stockage, 47
clavier, 56
Click, événement, 344
CLng, fonction, 171
codage, 31
code, 31
aide, 146
commentaires, 135
couleurs, 138
exécuter, 146
écriture, 135
fenêtre, 89, 96, 98
mise en forme, 137
objets, 91
options, 99
stockage, 113
collection d'objets, 14, 19
For Each…Next, 189
indice, 19
méthode, 25
Column, propriété, 59

COM, 174
combinaison (touches), 373
ComboBox, contrôle, 277, 307, 316, 317, 352
  valeur, 308
CommandButton, contrôle, 280, 313, 314
commande (enregistrer), 31
commenter, code, 135, 137
comparer (chaînes), 236
compilation, 246
  erreurs de, 245
Complément automatique des instructions, option, 147
concaténation
  chaînes de caractères, 229
  variables, 156
conditions
  imbriquées, 195
  opérateur relationnel, 178
Const, mot-clé, 169
constantes, 168
  caractères, 232
  définition, 116
  Excel, 23
  intérêt, 173
  préfixes, 23
  type de données, 23
contrôles
  ajouter, 293
  aligner, 289
  centrer, 291
  CheckBox, 279, 308, 316, 360
  ComboBox, 277, 307, 308, 316, 317, 352
  CommandButton, 280, 313, 314
  copier/coller, 285
  focus, 312
  Frame, 278
  grouper, 292
  info-bulle, 306
  Label, 276, 348
  ListBox, 278, 308, 357

mise en forme, 287
MultiPage, 281, 308
OptionButton, 280, 308, 325, 360
placer, 283
raccouri, 324
ScrollBar, 282, 308, 323, 324, 360
sélectionner, 286
séparer, 292
SpinButton, 283, 308, 323, 362
supprimer, 287
TabStrip, 281
tabulation, 312
TextBox, 277, 308, 316, 317, 319, 350
ToggleButton, 280, 308
uniformiser, 290, 291
valeur, 307
ControlTipText, propriété, 306
conventions, 8
copier/coller, 285
Cos, fonction, 223
couleurs du code, 138
CreateObject, fonction, 165, 173
créer
macro, 44
procédure, 132
CSng, fonction, 171
CStr, fonction, 171
CurDir, instruction, 447
Currency, type de donnée numérique, 155
CurrentRegion, propriété, 69
curseurs, 372
Cursor, propriété, 372
CVar, fonction, 171

D

Date
fonction, 193, 225
variable, 157
DateSerial, fonction, 226

DateValue, fonction, 226
Day, fonction, 225
DDB, fonction, 226
Deactivate, événement, 376
débogage, 245, 246
  espions, 252
  exemple, 255
  fenêtre Exécution, 252
  interroger, 249
  modifier, 251
  pas à pas, 248
  pile des appels, 254
  points d'arrêt, 250
  tester, 246
déclaration
  explicite et implicite, 149
  module, 93
  procédures, 122, 125, 127
  variables, 149
Default, propriété, 314
Delay, propriété, 322
déplacements
  codage VB, 59
  Excel, 55
désactivation, 376
détecteur d'erreur, 261
Dialog
  collection, 209
  objet, 209
Dim, mot-clé, 152, 172
Dir, instruction, 447
DisplayAlerts, propriété, 372
DisplayFormulaBar, propriété, 372
DisplayScrollBars, propriété, 372
DisplayStatusBar, propriété, 372
Do…Loop, structure de contrôle, 181
document hôte, 46
données
  conversion, 169, 171

validation, 169
dossiers, 78
Feuilles, 78
manipuler, 447
Microsoft Excel Objets, 78
Modules, 78
Références, 78
Double
clic, 376
type de donnée numérique, 155
DropButtonClick, événement, 344
durée de vie (variables), 172

E

écriture du code, 135
EnableCancelKey, propriété, 372
Enabled, propriété, 314
End
mot-clé, 145, 249
propriété, 69
End Sub
instruction, 37
enregistrer
commandes, 31
macro, 31, 32, 33
sous, 209
Enter, événement, 344
EnterKeyBehavior, propriété, 316
erreurs
compilation, 245
détecteur, 261
exécution, 246
gestion, 245, 261
gestionnaire, 261
logiques, 246
objet Err, 262
On Error, 261
espaces, 233
espions, 252

express, 254
supprimer, 254
étiquettes, 261
événements, 26, 92, 93, 343
Activate, 376
AfterUpdate, 343
BeforeDoubleClick, 376
BeforeDragOver, 343
BeforeDropOrPaste, 343
BeforeRightClick, 376
BeforeUpdate, 343
Calculate, 376
Change, 344, 376
Click, 344
Deactivate, 376
DropButtonClick, 344
Enter, 344
Exit, 344
feuille de calcul, 376
gestion des… utilisateurs, 369
Initialize, 344
KeyDown, 345
KeyPress, 345
KeyUp, 345
MouseDown, 347
MouseMove, 347
MouseUp, 347
procédures, 93, 115
SelectionChange, 376
SpinDown, 347
SpinUp, 347
Excel
barre, 372
boîtes de dialogue, 209
cellule active, 57
intitulé, 372
message, 372
modèle d'objets, 27
référence, 58, 60, 67

sélection, 56
exécution
code, 146
fenêtre, 252
impossible, 249
macro, 35
pas à pas, 248
Exit
événement, 344
mot-clé, 144, 261
Exp, fonction, 223
Explorateur d'objets, 77, 80
afficher, 80
aide, 83
fonctions, 219
masquer, 80
rechercher, 85
utiliser, 81
Explorateur de projet, 76, 77
afficher, 77, 79
masquer, 77
utiliser, 78
exporter (module), 130, 383
expressions, 116
arithmétiques, 22
opérateurs relationnels, 178

F

False, valeur booléenne, 22
fenêtre
ancrer, 108
Code, 76, 89, 91, 94, 96, 98, 99
Exécution, 252
Propriétés, 76, 101, 102
UserForm, 76, 86, 89
Variables locales, 249
feuille, 273
afficher, 298
contrôles, 102, 283

création et mise en forme, 273
créer, 274
de calcul, 372, 376
développement, 273
dossier, 78
masquer, 298
mettre en forme, 287
position, 330
raccourci, 324
fichiers, 447
FileCopy, instruction, 448
FileDateTime, instruction, 448
FileLen, instruction, 448
Filter, fonction, 241
Fix, fonction, 223
focus, 312
fonctions, 26
Abs, 223
arguments, 27
Asc, 224
Atn, 223
CBool, 171
CByte, 171
Ccur, 171
CDate, 171, 223
CDbl, 171
CDec, 171
Chr, 200, 224, 231
CInt, 171
CLng, 171
conversion, 171
Cos, 223
CreateObject, 165, 173
CSng, 171
CStr, 171
CVar, 171
Date, 193, 225
DateSerial, 226
DateValue, 226

Day, 225
DDB, 226
Excel, 217
Exp, 223
Filter, 241
Fix, 223
Format, 224
FV, 226
GetObject, 164, 173
Hex, 224
Hour, 225
InputBox, 199
InStr, 190
IPmt, 227
IRR, 227
IsArray, 170
IsDate, 170
IsEmpty, 170
IsError, 170
IsMissing, 170
IsNull, 170
IsNumeric, 170
IsObject, 170
Join, 231
LBound, 162
LCase, 235
Left, 190, 233
Len, 233
Log, 223
LTrim, 233
Mid, 233
Minute, 225
MIRR, 227
Month, 225
MsgBox, 204
Now, 225
NPer, 227
NPV, 228
Oct, 224

personnalisées, 218
Pmt, 227
PPmt, 227
PV, 228
Rate, 227
recommandations, 221
Replace, 234
Resize, 69
RGB, 224
Right, 233
Rnd, 223
Round, 223
RTrim, 233
Second, 225
Sgn, 223
Sin, 223
SLN, 226
Sqt, 223
Str, 224
StrComp, 236
StrConv, 235
String, 232
SYD, 226
Tan, 223
Time, 225
Timer, 226
TimeSerial, 226
TimeValue, 226
Trim, 233
TypeName, 170
types de données, 170
UBound, 162
UCase, 235
Val, 224
VarType, 170
VBA, 222, 223, 224, 225, 226
Weekday, 225
WeekdayName, 225
Year, 225

Font, propriété, 335
For Each…Next, structure de contrôle, 189
For…Next, structure de contrôle, 185
ForeColor, propriété, 306
Format, fonction, 224
Frame, contrôle, 278
FullName, instruction, 448
Function, procédures, 122
  appels, 142
  syntaxe, 122
FV, fonction, 226

G

gestion avancée des objets Excel, 369
gestion des erreurs, 245, 246
  exemple, 262
  gestionnaire, 261
GetAttr, instruction, 448
GetObject, fonction, 164, 173, 174
GetOpenFilename, méthode, 209
GetSaveAsFilename, méthode, 209
GoTo, instruction, 198
grille, 287
GroupName, propriété, 325

H

Height, propriété, 330
HelpContextID, propriété, 325
Hex, fonction, 224
HideSelection, propriété, 316
hiérarchie de classes, 15
Hour, fonction, 225

I

If…Then…Else, structure de contrôle, 193
Info express automatique, option, 148
info-bulles, 249, 306
Initialize, événement, 344
InputBox

fonction, 199
méthode, 202
instanciation, 14
d'un objet, 14
InStr, fonction, 190
instructions
casse, 45
ChDir, 447
ChDrive, 447
commenter, 137
complément, 147
conditionnelles, 192, 193, 197
CurDir, 447
d'affectation, 117
de déclaration, 116
Dir, 447
End Sub, 37
exécutables, 117
FileCopy, 448
FileDateTime, 448
FileLen, 448
FullName, 448
GetAttr, 448
GoTo, 198
imbriquées, 138
MkDir, 448
modifier, 251
Name, 448
présentation, 116
retraits de ligne, 137
RmDir, 448
SetAttr, 448
Sub, 37
sur plusieurs lignes, 135
Int, fonction, 223
Integer, type de donnée numérique, 155
interface de Visual Basic Editor, 76
interruption d'une macro, 372
intitulé, 276

barre de titre, 372
IPmt, fonction, 227
IRR, fonction, 227
Is, opérateur relationnel, 178
IsArray, fonction, 170
IsDate, fonction, 170
IsEmpty, fonction, 170
IsError, fonction, 170
IsMissing, fonction, 170
IsNull, fonction, 170
IsNumeric, fonction, 170
IsObject, fonction, 170

J

Join, fonction, 231

K

KeepScrollsVisible, propriété, 322
KeyDown, événement, 345
KeyPress, événement, 345
KeyUp, événement, 345

L

Label, contrôle, 276, 348
LargeChange, propriété, 324
LBound, fonction, 162
LCase, fonction, 235
Left
fonction, 190, 233
propriété, 330
libérer une variable objet, 167
Like, opérateur relationnel, 178
ListBox, contrôle, 278, 357
valeur, 308
liste déroulante, 278
Locked, propriété, 317
Log, fonction, 223
Long, type de donnée numérique, 155
LTrim

fonctions, 233
Len, 233

M

macros
accéder, 46, 52
améliorer, 41
autorisations, 384
complémentaires, 48, 49, 50
enregistrement, 32, 33, 58
enregistreur, 31
exécution, 35, 373
écran, 372
écrire, 44
globales, 47
instructions, 37
interrompre, 151, 372
sauvegarde, 383
signer, 395
stockage, 46
structure, 36
utilisation, 31
virus, 379
majuscules (convertir), 235
manipuler, 447
masquer, 298
matrice, variables (de), 159
Max, propriété, 323
MaxLength, propriété, 317
membres d’une classe, 80, 82
messages d’alerte, 372
méthodes, 19, 25
Application, 373
Calculate, 373
collection, 25
InputBox, 202
OnKey, 373
OnTime, 373
OnUndo, 373

Microsoft Excel Objets, dossier, 78
Mid, fonctions, 233
Min, propriété, 323
minuscules (convertir), 235
Minute, fonction, 225
MIRR, fonction, 227
mise à jour de l'écran, 372
mise en forme
  code, 137
  contrôles, 287
MkDir, instruction, 448
mode Arrêt, 249
modèle d'objets, 15, 18, 27
modifier
  bouton, 42
  cellule, 25
  chaînes, 229, 235
  Change, 344, 376
  commandes, 3
  contrôle, 296
  instructions, 251
  macro, 36
  objet, 20, 24
  propriétés, 104
  Resize, 60, 69
  SelectionChange, 376
  variable, 171
module
  affichage du code, 99
  ajouter, 129
  de classe, 78
  déclarations générales, 93
  dossier, 78
  exporter, 130, 383
  objets, 91
  présentation, 113
  procédures, 92
  supprimer, 130
Month, fonction, 225

mot de passe, 303
mot-clé, 38, 116
  As, 120
  ByRef, 120
  ByVal, 120, 121
  Call, 141
  Const, 169
  définition, 116
  Dim, 152, 172
  End, 145, 249
  Exit, 144, 261
  Nothing, 167
  On Error, 261
  Optional, 120
  ParamArray, 120
  Private, 119, 134, 172
  Public, 119, 134, 173
  REM, 136
  Resume, 261
  Set, 163
  Static, 119, 121, 173
  Stop, 145, 249, 251
  Type, 167
  WithEvents, 370
MouseDown, événement, 347
MouseIcon, propriété, 326
MouseMove, événement, 347
MousePointer, propriété, 326
MouseUp, événement, 347
MSForms, bibliothèque d'objets, 81
MsgBox, fonction, 204
MultiLine, propriété, 317
MultiPage, contrôle, 281
  valeur, 308

N

Name
  instruction, 448
  propriété, 302

nom d’utilisateur, 372
Nothing, mot-clé, 167
Now, fonction, 225
NPer, fonction, 227
NPV, fonction, 228

O

objets
  accéder, 18
  affecter une macro, 269
  aide, 83
  Application, 17, 369, 370, 372, 373
  bibliothèque, 16
  caractéristiques, 14
  classes, 14
  classeur, 372
  code, 91
  collections, 14, 19, 189
  définition (propriétés), 39
  Dialog, 209
  explorer, 80
  événements, 26
  gestion, 369, 377
  instanciation, 14
  méthodes, 25
  modèles, 15
  présentation (des), 13
  propriétés, 20
  recherche, 85
  référencer, 174
  référentiel, 19, 24
  ThisWorkbook, 374
  variables, 163
  Workbook, 19
  Worksheet, 376
Oct, fonction, 224
Office
  applications hôtes, 16
  bibliothèque, 81

Offset, propriété, 60
OLE Automation, 174
On Error, mot-clé, 261
onglet, 281
OnKey, méthode, 373
OnTime, méthode, 373
OnUndo, méthode, 373
opérateurs
  arithmétiques, 156
  comparaison, 178
  concaténation, 153
  logiques, 213
  relationnels, 178
option
  ancrage des fenêtres, 109
  Base 1, instruction, 159
  Explicit, option, 150
Optional, mot-clé, 120
OptionButton, contrôle, 280, 325, 360
  valeur, 308
Or, opérateur logique, 213
ordre de tabulation, 312, 328
  définir, 328
ouvrir
  boîte de dialogue, 209
  Visual Basic Editor, 73

P

ParamArray, mot-clé, 120
paramètres
  fonctions, 27
  Visual Basic Editor, 108
Pas à pas, commande, 248
passage d'arguments, 114, 119, 142
  apparition, 142
  nommés, 143
  référence, 120
  valeur, 120, 121
PasswordChar, propriété, 303

PathSeparator, propriété, 372
PERSONAL.XLSB, 47
personnaliser
  boîte à outils, 293
  boutons, 268
  raccourcis clavier, 265
Picture, propriété, 332
PictureAlignment, propriété, 333
PicturePosition, propriété, 334
PictureSizeMode, propriété, 334
PictureTiling, propriété, 334
pile des appels, 250, 254
plage de cellules
  courantes, 69
  fin, 69
  nommées, 71
  redimensionner, 69
  utilisées, 69
Pmt, fonction, 227
points d'arrêt, 250
portée
  des procédures, 119
  notion, 134
  variables, 172, 173
PPmt, fonction, 227
Private, mot-clé, 119, 134, 172
procédures
  appels, 114, 140
  créer, 132
  définition, 91
  déplacer, 140
  événement, 371, 374, 376
  événementielles, 115, 337
  Function, 122, 142
  noms, 119
  pas à pas, 248
  passage d'arguments, 142
  pile des appels, 250
  portées, 119, 134

présentation, 114
privées, 119, 134
Property, 124, 142
Property Get, déclaration, 125
Property Let, déclaration, 127
publiques, 119, 134
quitter, 144
récursives, 208
Sub, 118, 140
tester, 246
types de…, 118
programmation orientée objet, 13, 27
programme VBA
instructions, 116
lisibilité, 140
modules, 113
pas à pas, 248
procédures, 114
quitter, 145
structure, 113
projets
compiler, 246
dossiers constitutifs, 78
Explorateur de projet, 79
explorer, 77
réinitialiser, 247
structure, 129
tester, 246
verrouiller, 384
Property Get, procédures
appel, 126
syntaxe, 125
Property Let, procédures
appel, 128
syntaxe, 127
Property, procédures, 124
appels, 142
propriétés, 20
accéder, 24

Accelerator, 324
ActiveCell, 59, 372
ActiveSheet, 372
ActiveWorkbook, 372
Alignment, 303
Application, 372
AutoSize, 311
AutoTab, 312
AutoWordSelect, 313
BackColor, 304
BackStyle, 304
BorderColor, 305
BorderStyle, 305
Cancel, 313
Caption, 305, 372
Cells, 59
Column, 59
contrôles, 301
ControlTipText, 306
CurrentRegion, 69
Cursor, 372
Default, 314
Delay, 322
des contrôles, 102
DisplayAlerts, 372
DisplayFormulaBar, 372
DisplayScrollBar, 372
DisplayStatusBar, 372
en lecture seule, 20
en lecture-écriture, 20
EnableCancelKey, 372
Enabled, 314
End, 69
EnterKeyBehavior, 316
fenêtre, 101
Font, 335
ForeColor, 306
GroupName, 325
Height, 330

HelpContextID, 325
HideSelection, 316
KeepScrollsVisible, 322
LargeChange, 324
Left, 330
Locked, 317
Max, 323
MaxLength, 317
Min, 323
modifier, 104
MouseIcon, 326
MousePointer, 326
MultiLine, 317
Name, 302
Offset, 60
PasswordChar, 303
PathSeparator, 372
Picture, 332
PictureAlignment, 333
PicturePosition, 334
PictureSizeMode, 334
PictureTiling, 334
Range, 59
Row, 59
ScreenUpdating, 372
ScrollBar, 321
Select, 60
Selection, 60
SelectionMargin, 318
SmallChange, 323
SpecialEffect, 306
StartUpPosition, 330
Style, 307, 318
TabIndex, 328
TabKeyBehavior, 319
TabStop, 329
TextAlign, 319
ThisWorkbook, 372
Top, 330

TripleState, 319
UsedRange, 69
valeurs (des), 21, 39
Value, 307
Visible, 309
Width, 330
WordWrap, 320
Public, mot-clé, 119, 134, 173
PV, fonction, 228

R

Raccourcis clavier
déplacer, 56
macro, 265
Range, propriété, 59
Rate, fonction, 227
rechercher
chaîne, 237
fenêtre Code, 94
texte, 85
redimensionner (plage), 69
référence
absolue, 58, 60
boîte de dialogue, 174
dossiers, 78
relative, 58, 67
référentiel d'objet, 19, 24
réinitialiser, 247
REM, mot-clé, 136
remplacer
chaîne, 234
texte, 94, 96
réorganiser (boutons), 292
Replace, fonction, 234
Resize, fonction, 69
Resume, mot-clé, 261
retrait
automatique, option, 138
bouton, 138

ligne (code), 137
RGB, fonction, 224
Right, fonctions, 233
RmDir, instruction, 448
Rnd, fonction, 223
Round, fonction, 223
Row, propriété, 59
RTrim, fonctions, 233

S

sauvegarder (macros), 383
ScreenUpdating, propriété, 372
ScrollBar, 372
contrôle, 282, 308, 323, 324, 360
propriété, 321
Second, fonction, 225
sécurité, 379
macros, 379, 383, 384, 395
niveaux, 380
verrouiller, 384
Select Case, structure de contrôle, 197
Select, propriété, 60
sélection
cellules, 59, 60, 62, 376
Excel, 55
outil, 276
plusieurs contrôles, 286
Selection, propriété, 60
SelectionChange, événement, 376
SelectionMargin, propriété, 318
Set, mot-clé, 163
SetAttr, instruction, 448
Sgn, fonction, 223
signature numérique, 395
Sin, fonction, 223
Single, type de donnée numérique, 155
SLN, fonction, 226
SmallChange, propriété, 323
sortie

procédure, 144
programme, 145
souris, 57
SpecialEffect, propriété, 306
SpinButton, contrôle, 283, 323, 362
valeur, 308
SpinDown, événement, 347
SpinUp, événement, 347
Sqr, fonction, 223
Standard, 106
StartUpPosition, propriété, 330
Static, mot-clé, 119, 121, 173
stockage
macros, 46
Stop, mot-clé, 145, 249, 251
Str, fonction, 224
StrComp, fonction, 236
StrConv, fonction, 235
String
fonction, 232
type de données, 21
variables, 153
structure
macros, 36
programmes VBA, 113
projets, 129
With…End With, 42
structures de contrôle, 177
Boîtes de dialogue, 198
boucles, 177, 178, 181, 185, 189
GoTo, 198
instructions conditionnelles, 192, 193, 197
Opérateurs logiques, 198
Style, propriété, 307, 318
Sub
instruction, 37
procédures, 118, 140
supprimer
contrôles, 287

espaces, 233
module, 130
SYD, fonction, 226
syntaxe
accès, 19, 24
arguments, 119, 143
Const, 169
couleurs, 138
CreateObject, 165
Dim, 152
Do…Loop, 181
For Each…Next, 189
For…Next, 185
Function, 122
gestion des erreurs, 261
GetObject, 164
GoTo, 198
If…Then…Else, 193
InputBox, 199, 202
méthodes, 25
MsgBox, 204
Private, 172
procédures, 141, 337
Property Get, 125
Property Let, 127
Public, 173
Select Case, 197
Set, 163
Static, 173
Type, 167
vérification automatique, 146
While…Wend, 178
With…End With, 39

T

TabIndex, propriété, 328
TabKeyBehavior, propriété, 319
tableaux, 120
instruction Option Base, 159

TabStop, propriété, 329
TabStrip, contrôle, 281
Tag, propriété, 329
Tan, fonction, 223
TextAlign, propriété, 319
TextBox, contrôle, 277, 316, 317, 319, 350
  valeur, 308
ThisWorkbook
  objet, 374
  propriété, 372
Time, fonction, 225
Timer, fonction, 226
TimeSerial, fonction, 226
TimeValue, fonction, 226
ToggleButton, contrôle, 280
  valeur, 308
Top, propriété, 330
touches, combinaison, 373
trait de soulignement, 135
Trim, fonctions, 233
TripleState, propriété, 319
True, valeur booléenne, 22
Type, mot-clé, 167
TypeName, fonction, 170
types de données, 153
  chaîne de caractères, 21
  constantes, 23
  conversion, 171
  valeur, 21, 22
  validation, 169
  vérifier, 169

U

UBound, fonction, 162
UCase, fonction, 235
Underscore, 135
uniformiser
  espace, commande, 291
  taille, commande, 290

UsedRange, propriété, 69
UserForm, fenêtre, 86

V

Val, fonction, 224
valeurs
  booléennes, 22, 157
  des contrôles, 307
  numériques, 21, 155
Value, propriété, 307
variables, 149
  boolean, 157
  concaténation, 156
  conversion du type, 171
  Dates, 157
  de matrice, 240, 241
  déclarer, 149
  définir, 170
  durée de vie, 172
  forcer la déclaration, 150
  info-bulles, 249
  locales, 121, 249
  matrice, 159, 231
  numériques, 155
  objets, 163, 167
  personnalisées, 167
  portées, 172
  statiques, 121, 173
  String, 153, 154
  types, 153
  validation, 169
  Variant, 158
  vérifier, 169
Variant, variables, 158
VarType, fonction, 170
VBA, bibliothèque d'objets, 82
vérification (syntaxe), 146
verrouiller un projet, 384
virus, macros, 379

Visible, propriété, 309
Visual Basic
  mots-clés, 38, 116
  présentation, 13
  rechercher, 85
  structure, 113
Visual Basic Editor
  ancrer les fenêtres, 108
  barres d'outils, 105
  développer dans, 113
  environnement, 73, 76
  Explorateur, 77, 80
  fenêtre, 86, 89, 101
  lancer, 73
  paramétrer, 108
  quitter, 75

W

Weekday, fonction, 225
WeekdayName, fonction, 225
While…Wend, structure de contrôle, 178
Width, propriété, 330
With…End With, structure, 39, 42
WithEvents, mot-clé, 370
WordWrap, propriété, 320
Workbook
  événements, 374
  objets, 19, 374
Worksheet
  événements, 376
  objets, 376

X

Xor, opérateur logique, 213

Y

Year, fonction, 225

Z

zone de liste, 277
zone de texte, 277

Pour suivre toutes les nouveautés numériques du Groupe Eyrolles, retrouvez-nous sur Twitter et Facebook

@ebookEyrolles

EbooksEyrolles

Et retrouvez toutes les nouveautés papier sur

@Eyrolles

Eyrolles